

Modelling and Generating Strategy Games Mechanics



Tobias Mahlmann

Center For Computer Games Research

IT University of Copenhagen

A thesis submitted for the degree of

Doctor of Philosophy

March 2013

Abstract

Strategy games are a popular genre of games with a long history, originating from games like *Chess* or *Go*. The word “strategy” is derived from the Greek word στρατηγός (strategós), meaning (the) general, resp. στρατηγεία, meaning “art of commanding”. Both terms directly hint at the subject of strategy games: warfare. Players must utilise armies and resources to their advantage on a battlefield to win a “virtual war”. The first strategy games were published as “Kriegsspiele” (engl. wargames) in the late 18th century, intended for the education of young cadets. Since then strategy games were refined and transformed over two centuries into a medium of entertainment. Today’s computer strategy games have their roots in the board- and roleplaying games of the 20th century and enjoy great popularity. In this thesis, we use strategy games as an application for the procedural generation of game content.

Procedural game content generation has regained some interest in recent years, both among the academic community and game developers alike. When the first commercial computer games were published in the early 1980s, technical limitations prevented game developers from shipping their titles with a large amount of pre-designed game content. Instead game content such as levels, worlds, stories, weapons, or enemies needed to be generated at program runtime to save storage space and memory resources. It can be argued that the generation of game content “on the fly” has gained popularity again for two reasons: first, game production budgets have risen rapidly in the past ten years due to the necessary labour to create the amount of game content players expect in games today. Secondly: the potential audience for games grew in the past years, with a diversification of player types at the same time. Thus game developers look for a way to tailor their games to individual players’ preferences by creating game content adaptively to how the player plays (and likes) a game.

In this thesis we extend the notion of “procedural game content generation”

by “game mechanics”. Game mechanics herein refer to the way that objects in a game may interact, what the goal of the game is, how players may manipulate the game world, etc. We present the Strategy Games Description Language (SGDL) and its adjacent framework, a tree-based approach to model the game mechanics of strategy games. The SGDL framework allows game designers to rapid prototype their game ideas with the help of our customisable game engine. We present several example games to demonstrate the capabilities of the language and how to model common strategy game elements. Furthermore, we present methods to procedurally generate and evaluate game mechanics modelled in SGDL in terms of enjoyability. We argue that an evolutionary process can be used to evolve the mechanics of strategy games using techniques from the field of machine learning. Our results show that automated gameplay combined with expert knowledge can be used to determine the quality of gameplay emerging from game mechanics modelled in SGDL, and that algorithms can augment the creativity of human game designers.

Acknowledgements

I would like to acknowledge all the individuals who provided me with insight and inspiration, in the form of discussions or collaborations, and contributed to my work and this thesis. Especially mentioned here should be my supervisors Julian Togelius and Georgios N. Yannakakis and all the colleagues whom I met at the Center for Computer Games Research at the ITU Copenhagen.

Contents

| | |
|---|-----------|
| List of Figures | ix |
| List of Tables | xi |
| Overview | i |
| 1 Introduction | 1 |
| 1.1 Computational Intelligence in Games | 4 |
| 1.2 Published papers | 7 |
| 2 Strategy Games | 9 |
| 2.1 The History of Strategy Games | 13 |
| 2.2 Strategy Games: The Delimitation of the Genre | 17 |
| 3 Related Theoretical Frameworks | 23 |
| 3.1 Modelling Player Experience: Qualitative Methods | 23 |
| 3.1.1 Self-reported data | 25 |
| 3.1.2 Requirements elicitation | 26 |
| 3.1.3 Personality | 29 |
| 3.1.4 Flow | 29 |
| 3.1.5 Emotional State | 31 |
| 3.1.6 Immersion | 33 |
| 3.2 Modelling Player Experience: Quantitative Methods | 34 |
| 3.2.1 Interest | 34 |
| 3.2.2 Physiological measures | 36 |
| 3.2.3 Tension in Board Games | 37 |
| 3.2.4 Measures from Combinatorial Games | 38 |

CONTENTS

| | | |
|----------|---|-----------|
| 3.2.5 | Learnability | 42 |
| 3.3 | Defining Game Mechanics | 44 |
| 3.3.1 | Different Layers of Rules | 45 |
| 3.3.2 | Games as Systems | 46 |
| 3.4 | Summary | 47 |
| 4 | Related computational intelligence in games research | 49 |
| 4.1 | Generating Content | 49 |
| 4.1.1 | Search-based procedural content creation | 49 |
| 4.1.2 | Procedural Level Generation | 51 |
| 4.1.3 | Interactive Storytelling | 51 |
| 4.1.4 | Cellular automata | 52 |
| 4.1.5 | L-Systems | 53 |
| 4.2 | Computational Creativity | 54 |
| 4.3 | AI and Learning in Games | 56 |
| 4.3.1 | Game Tree Search | 56 |
| 4.3.1.1 | Min-Max Search | 58 |
| 4.3.1.2 | Monte-Carlo Tree Search | 59 |
| 4.3.2 | State machines | 62 |
| 4.3.3 | Decision- and Behaviour Trees | 63 |
| 4.3.4 | Neural Networks | 63 |
| 4.3.5 | Genetic Algorithms | 64 |
| 4.3.6 | Genetic Programming | 65 |
| 4.4 | Modelling and Generating Game Mechanics | 66 |
| 4.4.1 | Unified Modelling Language (UML) | 66 |
| 4.4.2 | Stanford GDL | 67 |
| 4.4.3 | Answer Set Programming | 68 |
| 4.4.4 | Ludi | 69 |
| 4.4.5 | ANGELINA | 70 |
| 4.4.6 | Fixed length genome | 71 |
| 4.5 | Summary | 71 |

| | | |
|----------|--|------------|
| 5 | The Strategy Games Description Language (SGDL) | 73 |
| 5.1 | Design Paradigms | 73 |
| 5.2 | Basic Concepts and Terms | 75 |
| 5.3 | The SGDL Tree | 77 |
| 5.3.1 | Attributes and Object References | 78 |
| 5.3.2 | Conditions | 78 |
| 5.3.3 | Consequences | 82 |
| 5.3.4 | Actions | 82 |
| 5.3.5 | Multiple Consequences | 83 |
| 5.3.6 | Action as Consequences | 85 |
| 5.3.7 | Object Lists and Object Filter | 85 |
| 5.3.8 | Winning Conditions | 87 |
| 5.4 | Surrounding framework | 87 |
| 5.4.1 | Maps | 88 |
| 5.4.2 | Game- and Player State | 90 |
| 5.5 | Comparison to other Game Description Languages | 91 |
| 5.6 | Summary | 93 |
| 6 | SGDL in Practice | 95 |
| 6.1 | Example Games | 95 |
| 6.1.1 | Simple Rock Paper Scissors | 95 |
| 6.1.2 | Complex Rock Paper Scissor | 96 |
| 6.1.3 | Rock Wars | 98 |
| 6.1.4 | Dune 2 | 102 |
| 6.2 | Interlude: “Spicing up map generation” | 102 |
| 6.3 | Summary | 106 |
| 7 | Automated Gameplay | 109 |
| 7.1 | The game, the agents and the assigner | 110 |
| 7.2 | Agents | 113 |
| 7.2.1 | Random action selection | 114 |
| 7.2.2 | Finite-state machine | 115 |
| 7.2.3 | Neuroevolution of augmenting topologies | 116 |
| 7.2.4 | MinMax | 117 |

CONTENTS

| | | |
|-----------|--|------------|
| 7.2.5 | Monte Carlo Tree Search | 119 |
| 7.2.6 | Potential fields | 120 |
| 7.2.7 | Classifier systems | 121 |
| 7.3 | Results of agent versus agent training | 122 |
| 7.4 | Results of human play testing | 127 |
| 7.5 | Conclusions | 131 |
| 8 | Measuring Game Quality | 133 |
| 8.1 | A Definition of Balance | 135 |
| 8.1.1 | Complementary Unit Types | 137 |
| 8.1.2 | Fitness function | 140 |
| 8.2 | The Outcome Uncertainty of a Game | 142 |
| 8.3 | Avoiding a Start-Finish victory | 144 |
| 8.4 | Fitness functions for Action subtrees | 146 |
| 9 | Searching the Strategy Game Space | 151 |
| 9.1 | Balance and Complementary Unit Sets | 152 |
| 9.2 | Comparing balance to outcome uncertainty | 155 |
| 9.2.1 | Evolving actions | 158 |
| 9.2.2 | The solution representation | 159 |
| 9.2.3 | The Evolutionary Algorithm | 160 |
| 10 | Overall Results | 163 |
| 10.1 | Further experiments | 168 |
| 10.2 | Game in focus: a qualitative analysis | 171 |
| 10.3 | Summary | 173 |
| 11 | Beyond tactics: Conclusions and Discussion | 175 |
| 11.1 | Modelled Games | 175 |
| 11.2 | New Game Mechanics | 176 |
| 11.3 | The Strategy Games Description Language | 177 |
| 11.3.1 | Verbosity vs. Versatility | 178 |
| 11.4 | The SGDL Framework | 179 |
| 11.5 | Computational Complexity | 180 |
| 11.5.1 | Status of the General Gameplaying Agents | 180 |

CONTENTS

| | |
|--------------------------------------|------------|
| 11.6 Human studies | 181 |
| 11.7 Data Driven Evolution | 182 |
| 11.8 Lessons learned | 183 |
| 11.9 Concluding Remarks | 185 |
| References | 187 |

CONTENTS

List of Figures

| | | |
|------|--|----|
| 3.1 | Illustration of the concept of flow | 30 |
| 4.1 | Game Tree of an arbitrary two-player game | 57 |
| 4.2 | The Mini-Max tree algorithm | 59 |
| 4.3 | Example of a state machine | 62 |
| 4.4 | A GP tree modelling the example $y = 3x^2 + 4x + 12$ | 66 |
| 5.1 | Selecting an Action in the SGDL Game Engine | 77 |
| 5.2 | The overall SGDL tree. | 79 |
| 5.3 | Overview and graphical representation of the SGDL node types. | 79 |
| 5.4 | Accessing a property | 80 |
| 5.5 | Two elementary examples of subtrees comparing constants. | 80 |
| 5.6 | Extended conditions | 81 |
| 5.7 | A single assignment as a simple Consequence. | 82 |
| 5.8 | A simple Action with one Condition and one Consequence | 83 |
| 5.9 | Side effects of multiple attribute changes | 84 |
| 5.10 | Ordered Consequences can be denominated with indexed vertices. | 85 |
| 5.11 | Actions as Consequences | 86 |
| 5.12 | Examples for using an ObjectList. | 87 |
| 5.13 | Two examples of winning conditions | 88 |
| 5.14 | Three ways of accessing a WorldObject on a map. | 89 |
| 5.15 | Examples of using the _DISTANCE SpecialFunction | 92 |
| 5.16 | Accessing a game state variable in an Action. | 92 |
| 6.1 | Subtree of “Class A” of “Simple Rock Paper Scissor” | 96 |
| 6.2 | The attack action in simple Rock Paper Scissor | 97 |

LIST OF FIGURES

| | | |
|------|---|-----|
| 6.3 | Screenshot of “Complex Rock Paper Scissors” | 98 |
| 6.4 | Adding range conditions to attack actions | 99 |
| 6.5 | Screenshot of “Rock Wars” | 100 |
| 6.6 | The detailed subtree for a “Create” action in “Rock Wars”. | 101 |
| 6.7 | Dune II in the SGDL engine. | 103 |
| 6.8 | Screenshot of the Dune II map generator application | 105 |
| 6.9 | Generated Dune II map examples | 107 |
| 7.1 | The Commander framework used for the agents in the study. | 111 |
| 7.2 | Finite-state automata of the SemiRandom agent units | 115 |
| 7.3 | Finite-state automaton of the FSM agent’s units | 115 |
| 7.4 | Action tree used to find MultiActions | 119 |
| 7.5 | Action tree illustrating the use of buildings. The building nodes are allowed only one child each to limit the complexity. | 119 |
| 7.6 | Summary of agent versus agent results | 123 |
| 7.7 | Summary of human play results | 129 |
| 8.1 | Rock-Paper-Scissors-Lizard-Spock | 137 |
| 8.2 | Screenshot from Warcraft III | 138 |
| 8.3 | Screenshot from Blue Byte’s <i>Battle Isle II</i> | 139 |
| 8.4 | Configurations for complementary unit sets in CRPS | 141 |
| 8.5 | Lead graph for an arbitrary two player game | 145 |
| 9.1 | Screenshot of the SGDL Editor prototype | 152 |
| 9.2 | The outcome uncertainty in <i>Rock Wars</i> | 155 |
| 9.3 | Overview of the evolutionary algorithm | 160 |
| 10.1 | An evolved Action | 165 |
| 10.2 | An evolved action adjusting min- and maximum ranges | 167 |
| 10.3 | Fitness development over 100 generations | 169 |
| 10.4 | Fitness developments over all five runs of the experiment using $\gamma =$ $0.5, \lambda = 0.5$ | 170 |

List of Tables

| | | |
|------|---|-----|
| 7.1 | Unit properties for SGDL models | 114 |
| 7.2 | Agents in the study | 114 |
| 7.3 | Summary of agent versus agent results | 124 |
| 7.4 | Standard deviantions of agent versus agent results | 125 |
| 7.5 | Results of the agent tournament | 126 |
| 7.6 | P-values for the human play results | 128 |
| 9.1 | A unit type set with fitness 0.0. | 153 |
| 9.2 | A unit type set with fitness 0.24. | 154 |
| 9.3 | A unit type set with fitness 0.57. | 154 |
| 9.4 | The correlation between different properties of sampled games of CRPS | 157 |
| 9.5 | The correlation between different properties of Rock Wars | 157 |
| 9.6 | Overview of the attributes of each class in CRPS | 159 |
| 10.1 | The weighting factors used in five different experiments. | 164 |
| 10.2 | Attributes of the classes found after 100 generations | 168 |

LIST OF TABLES

Overview

This thesis is organised as follows: first, I will try to motivate the research questions addressed in this thesis by outlining the historical development of procedural content generation in games and the enclosing academic field of computational intelligence in games in chapter 1. Chapter 2 will introduce a set of working delimiting properties of the term “strategy games” based on its historical connotations and common conventions used in games attributed to this genre. Chapter 3 and 4 will provide the reader with a list of related research divided into frameworks to measure the player’s experience (chapter 3) in general, and techniques specific to the problem of generating game content or artificial decision making (chapter 4). Chapter 5 will introduce the main contribution of this thesis: the strategy games description language (SGDL), a domain specific language to express the game mechanics of strategy games as laid out in chapter 2. Chapter 6 will then present some examples how SGDL may be set in practice by modelling some simple example games. The chapters 7, 8, and 9 will shift the focus to the automated generation of game rules of strategy games. Chapter 7 starts by introducing the agent framework used to play strategy games expressed in SGDL. Chapter 8 will provide the reader with a series of game quality measures used to determine the quality of entertainment certain strategy game mechanics would provide. Both, simulated gameplay and quality measures, are ultimately connected in chapter 9 in a series of experiments to demonstrate the capabilities of SGDL in the generation approach, presenting the results in chapter 10. The thesis concludes in chapter 11 with the discussion of the results, the potential of the approach, its shortcomings, and potential future directions of research.

0. OVERVIEW

Chapter 1

Introduction

During my years as a full time researcher I engaged in a lot of conversations about why we (as individuals) do research in games. I guess for me it is that “I always had a passion for games”. In fact I heard this answer from many colleagues, and apparently it got so over used by many of our MSc students that a senior colleague in my research group got seriously upset about it one year. But there is some truth to it: I remember my 10th birthday, when I was so fascinated about computer games that I decided to study computer science and do “something” with computer games for a living. The reason was simple: I had played through all the games I owned at that time, the internet as a source for new games wasn’t available yet, and with my little pocket money I wasn’t able to buy more games. Instead, I dreamt of making my own games, or even better: a system that would create games of my taste.

It would be far-fetched to say that I already foresaw the research described in the following chapters, but this little anecdote shall serve as a small scale allegory of what challenges exist today in the games industry. In the home computer era, the early 1980s, published games were normally novel in a sense that they presented new game-play ideas. Over the following decades games with similar elements formed genres, e.g. platform games or adventures, each of them with rising and falling popularity. Overall, the popularity of “gaming” has seen an enormous increase in the last twenty years. The question *why* is wide enough for another dissertation, and I can only speculate. However, what we could observe was a rapid technological development of hardware components such as central processors, graphics hardware, or gaming systems. Over the years more platforms became available: gaming consoles, hand-held devices, mo-

1. INTRODUCTION

mobile phones, and personal computers - perpetuating generations of games. And each generation of games tried to utilise the capabilities of the newest hardware generation available at maximal level. Until the early 2000s most game companies focused on improving the graphical quality of their games in terms of technical features, e.g. screen resolutions, number of polygons, or shader effects. On the other hand, the number of people interested in computer games increased massively in recent years. A blog post titled *Transformation of the Industry* in the Electronic Arts *Spielkultur*¹ blog (1) presented data, that in the year 2000 worldwide approx. 200 million people were considered as “gamers”, while their numbers climbed to 1.2 billion in the year 2012. Although these numbers are the result of Electronic Arts’ internal market research, and can’t be externally verified, they do seem believable. The German Trade Association of Interactive Entertainment Software (BIU) reports an increase of market volume between 2010 and 2011 by 3.5% to 1.99 billion Euro for Germany alone (2), based on a representative survey of 25,000 individuals. These numbers already illustrate that games are becoming the mainstream medium of the 21st century.

With more people playing computer- and video games, a larger potential market volume, and more diverse preferences due to the number of players, the industry faced a competition for the favour of the players, i.e. the paying customers. The appeal of a game to a wider range of players directly correlates with the game’s financial success. While the ground-breaking technological advances were the strongest diversifying criterion in the 1990s, most games published in the early 2000s had reached a certain industry standard in audio-visual quality. Today it is very rare that a professionally published game has technical deficiencies or outdated graphics, in a sense that it uses obsolete technologies. However, differentiation is still possible, and some titles seen as more technologically advanced than others. However, this does not include special art styles, e.g. abstract or cartoon-like, which are nowadays chosen deliberately. The standardisation of hard- and software systems played an important role in this process, e.g. application interfaces such as *DirectX* or *OpenGL* removed the necessity to implement special cases for a specific graphics card or sound processor in every game. Although the development of technology never stopped, technical advancements seemed to stagnate between generations of games in the past ten years. Game series like *FIFA Soccer* illustrate this very well: the graphical advancements compared to the direct predecessor

¹Freely translated: “culture of play”

of a game are rather cosmetic. Naturally, these observations are very subjective, but the development cycles of the mentioned application interfaces support this impression: until version 9, DirectX major versions were published in an annual cycle. The first version of revision 9 was released in 2002, the next major version 10 was published in 2006, and version 11 in 2009¹. Although minor releases and updates were published in-between, this indicates a consolidation in game technology throughout the past ten years. In this process developers shifted their focus to the *content* of a game. The word *content* here refers to all aspects of a game that affect gameplay but are not the game engine itself. This definition includes such aspects as terrain, maps, levels, stories, dialogue, quests, characters, etc. (3). Players expected a certain standard in game quality and more “value” in return for their purchase. In other words, players asked for games that had more levels, more stories, more characters, etc. A diversification also took place in the preferred game length (the time needed to finish a game): some players ask for a very long game, others prefer a short but intense experience. Nevertheless, the amount of content put in a game can be directly translated into man hours, as it takes more time to create more 3D-models or write more dialogue lines. The cost of computer games production has therefore increased drastically in the past years. Although it is hard to find reliable data on production costs, budgets between 10–40 million US Dollars for larger productions seem likely (4). Major productions such as *Grand Theft Auto IV* or *Star Wars: The Old Republic* even reported budgets of 100 million resp. 200 million US Dollars (5, 6). It can be said that budgets of the video game industry compete with those of major Hollywood film productions. But it should be added though, that on the other end of the budget scale are a multitude of small projects as well, realised by firms specialised on niche- or innovative games. But despite the number of games by smaller firms, financially successful examples such as *Minecraft* remain an exception. These firms are often confronted with the problem of reaching the industry standard with a small budget. They often rely on market shares that are not dominated by titles by the major publishers, or include innovative game elements into their games as a unique selling point. Of course, other factors like non-existing marketing budgets play a significant role as well. But overall, analogously to “alternative films”, a so called “Indie Game Scene”² enjoys a rising popularity (7). Ultimately,

¹All release dates are taken from <http://en.wikipedia.org/wiki/DirectX>

²from being “independent” from external investors

1. INTRODUCTION

both market segments are confronted with the challenge of efficiently creating content.

It seems therefore natural, that ideas which were already part of the earliest games recently regained some interest — but for different reasons: the technology of the 1980s did not permit game designers to put a large quantity of pre-designed game content into their games, as that would require more data to be stored on the game’s cartridge or floppy disc. The storage capacities of floppy discs, cassettes, cartridges, and RAM restricted the use of pre-designed game content to a minimum. Instead, game developers resorted to a different solution, the algorithmic creation of game content while the player was playing the game. Parts were created at the time they were needed. One of the earliest and prominent examples from 1984 was the space trading game *Elite*, in which the player had to fly from solar system to solar system and trade different goods or succeed in piracy and military missions. Whenever the player entered a new solar system, the program generated the available cargo and its prices at that very moment. This provided the illusion that the game simultaneously simulated all of the 2000 solar systems which were part of the game. Although today’s technology lifted these restrictions, content generation algorithms regained some interest due to increased demand for game content described in the previous paragraphs. Game developers are looking for more efficient ways to create game content than manual design.

1.1 Computational Intelligence in Games

From an academic point of view, I would like to direct the reader’s attention to an academic field which gained some importance in the past years, the *computational intelligence in games* (CIG) community. It is, besides *game analysis* (or *game studies*) on the humanist side and *game design* research, one of the columns forming games research. The field of CIG forms a link between games and classical computer science, such as software engineering, artificial intelligence research, machine learning, or algorithmic theory — just to name a few. I will discuss a multitude of applications related to my thesis in chapter 4. Coming back to the problem of content generation, part of the research published in the CIG community is within the sub-field of *procedural content generation* (PCG) which addresses the algorithmic generation of game content. Game content normally refers to the previously stated aspects of gameplay (e.g. weapons, textures, levels, or stories) and may — to distinguish PCG from other fields of research

— exclude any aspect connected to agent behaviour. It might be argued though, that generating behavioural policies might be considered PCG in some contexts. One particular approach to PCG which has gained traction in recent years is the *search-based* paradigm, where evolutionary algorithms or other stochastic optimisation algorithms are used to search spaces of game content for content artefacts that satisfy gameplay criteria (3). In search-based PCG, two of the main concerns are how this content is represented and how it is evaluated (the fitness function). The key to effective content generation is largely to find a combination of representation and evaluation such that the search mechanism quickly zooms in on regions of interesting, suitable, and diverse content. The major challenge in designing a fitness function is therefore to define what is *interesting* to the player. The field of PCG is therefore closely related to another CIG field, the discipline of *player experience modelling* (8, 9). There exist many theories describing what makes a game “fun” which originate from different disciplines and which were created independently in many cases. While they are useful for high-level (manual) game design, algorithmic creation of game content requires quantitative models. Empirical research created several models in the past decade (10), some of them will be used and presented throughout this thesis in chapter 3.

This thesis addresses a particular problem of search-based PCG, the procedural generation of game rules. Game rules define how things “work” in a game, how objects interact, which interaction possibilities the players have, and how to win the game etc. With the previously described players’ demand for more game content, game genres being well established, and budgets in the millions, publishers often reduce their financial risk by publishing successors of already established games and brands – relying on game ideas that have been proven to produce financially successful games. Like Hollywood “blockbusters”, these games do not need to be uninteresting by default, but one can often read the demand for new game ideas in community boards or media. The previously mentioned indie game scene can be seen as a symptom of this trend. Deterred by major publishers’ product development strategies, small firms or individuals often try to finance their projects with their own capital to retain full creative control. With lesser financial risks involved, game developers have the freedom of taking larger risks by implementing innovative game mechanics which eventually only appeal to a small audience; equivalent to a smaller return of investment.

1. INTRODUCTION

The genre I used as an application of procedural rule generation in this thesis is *strategy games*. Games that fall under this definition are titles such as *Civilization*, *Starcraft*, or *Battle Isle*. In this genre of games, the player often takes the role of a commander, general, or leader in general, facing challenges such as warfare, economic simulation, or diplomacy. The term “strategy games” will be better defined in section 2. I chose to work with this genre because - coming back to my opening anecdote - these were and are my favourite genre of games. But besides personal preference, these games provide an excellent application for various algorithms. It could be argued that they are probably the most complex games published so far, in a sense of how much information a player must consider for his next action. The complexity of strategy games provides a good challenge for procedural content generation techniques and agent design¹ alike. And to my knowledge no commercially available games use the dynamic generation of rules so far. In chapter 4 I will also discuss the academic projects with other game genres that have been proposed so far.

I structured this thesis as follows: in chapter 2 I will refine the definition of “strategy game” which will be used throughout this thesis. I will also discuss aspects of the historical context of strategy games. Chapters 3 and 4 will present relevant and related techniques, that have been published so far. The main part of this thesis will start with the presentation of the *Strategy Games Description Language* framework in sections 5 and 6, followed by the general gameplaying part of the framework in chapter 7. The framework was used in several experiments to measure the quality of strategy game mechanics. The applications and findings will be presented in chapters 8, 9, and 10. The thesis will conclude with an outlook of possible research directions based on the findings so far in chapter 11.

Although this thesis covers my own work, over the last three years at the ITU Copenhagen, many parts have been created and published in collaborations with other researchers. I will therefore shift the narrative point-of-view to the academic form of “we” in the following chapters.

¹agents that can act as an ally or enemy. In game contexts these are often called “bots”.

1.2 Published papers

Much of the work presented in this thesis has previously been published as peer-reviewed papers in conferences and other venues. The following is a list of the papers I published during my PhD, and which the thesis is partly based on:

1. Mahlmann, Tobias and Togelius, Julian and Yannakakis, Georgios N., **Towards Procedural Strategy Game Generation: Evolving Complementary Unit Types**, In *Applications of Evolutionary Computation*, 2011, pages 93–102, Springer Verlag
2. Mahlmann, Tobias and Togelius, Julian and Yannakakis, Georgios N., **Modelling and evaluation of complex scenarios with the Strategy Game Description Language**, In *Proceedings of the Conference on Computational Intelligence and Games (CIG)*, 2011
3. Mahlmann, Tobias and Togelius, Julian and Yannakakis, Georgios N., **Spicing up map generation**, In *Proceedings of the 2012 European conference on Applications of Evolutionary Computation*, 2012, EvoApplications’12, pages 224–233, Springer Verlag
4. Nielsen, Jon Lau and Fedder Jensen, Benjamin and Mahlmann, Tobias and Togelius, Julian and Yannakakis, Georgios N., **AI for General Strategy Game Playing**, In *IEEE Handbook of Digital Games*, 2012, Manuscript submitted for review
5. Mahlmann, Tobias and Togelius, Julian and Yannakakis, Georgios N., **Evolving Game Mechanics For Strategy Games**, In *Transactions on Computational Intelligence and AI in Games*, 2013, Manuscript submitted for review

1. INTRODUCTION

Note that the thesis provides more details on almost all aspects of the work presented in these papers. Additionally, I have contributed to the following papers, which are not included in the thesis:

1. Mahlmann, Tobias and Drachen, Anders and Canossa, Alessandro and Togelius, Julian and Yannakakis, Georgios N., **Predicting Player Behavior in Tomb Raider: Underworld**, In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, 2010, pages 178–185
2. Salge, Christoph and Mahlmann, Tobias, **Relevant Information as a Formalised Approach to Evaluate Game Mechanics**, In *Proceedings of the Conference on Computational Intelligence and Games (CIG)*, 2010
3. Mahlmann, Tobias and Togelius, J. and Yannakakis, Georgios N., **Evolving Card Sets Towards Balancing Dominion**, In *IEEE World Congress on Computational Intelligence (WCCI)*, 2012
4. Font, José M. and Mahlmann, Tobias and Manrique, Daniel and Togelius, Julian, **A card game description language**, In *Proceedings of the 2013 European conference on Applications of Evolutionary Computation*, 2013, Manuscript submitted for review

Chapter 2

Strategy Games

Before designing a modelling language for strategy games it seems beneficial to define what “strategy games” actually are. The boundaries between genres seem blurry, and no formal definition is known to the authors. We will therefore make an excursion into the field of media science and history to elicit a working definition of the genre of strategy games to have a clearer perspective on modelling requirements. This chapter will present a short history of the term “strategy” and its various meanings, continue with a brief overview of the history of strategy games as a medium, and conclude with a definition of a (computer) strategy game which we will work with throughout this thesis.

Strategy games have been used as a field of experiments by several publications before. It seems that they are a viable application for agent designs (11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21). The majority of the cited sources presents different techniques to use in a bot for the popular real-time strategy game *Starcraft*, among them Bayesian networks or potential fields. But also technical aspects such as network latency (22) or the effect of cheating (23) have been researched. Commonly, experiments are conducted using a commercially available game or a test-bed created specifically for that experiment, and the authors claim that the experiment sufficiently represents the whole genre of “strategy games”. Yet, publications in the field of computer science do not narrow the term and establish its boundaries. It seems therefore necessary establish a viable definition of the term “strategy game” for our research.

The word “strategy” is derived from the Greek word στρατηγός (strategós) for (the) general. A similar meaning comes from στρατηγεία which may be translated into the

2. STRATEGY GAMES

“art of commanding”. In other words, it stands for the systematic planning of achieving a certain (militaristic) goal. An early example of the definition of strategy games can be found in Harold Murray’s book about *The history of Chess*:

... [Chess] must be classed as a game of war. Two players direct a conflict between two armies of equal strength upon a field of battle, circumscribed in extent, and offering no advantage of ground to either side (24)

About four decades later, Roberts et al. extended this definition observing that the role of the two “armies” is quite complex, and the game actually models a complex social system (25). Roberts writes based on the work of Murray:

The role structure of the two “armies” is quite complex. Both the structure and the terminology of such a game of strategy may offer clues to the nature of the interactive system it represents. Pieces are called “men”; they capture or kill, they attack or defend, they race, and so on.

Roberts hypothesises further, that a social system can only be modelled as a game if it is complex enough. In return “simple societies should not possess games of strategy and should resist borrowing them.”

Strategy Games in Culture and Media If we consider Huizinga’s definition of a game, described in his book *Homo Ludens* (26), the word “strategy” seems directly opposed to Huizinga’s definition of “play” at first:

Summing up the formal characteristic of play, we might call it a free activity [...] It is an activity connected with no material interest, and no profit can be gained by it.

Rolf Nohr found this paradox in his disquisition on the series of papers “Strategie spielen” (to play strategy) (27). He concluded that the affinity between “strategy” and “play” becomes only visible when “strategy” is considered a “certain form of thinking” (ibid., p. 7). Strategic thinking is the prerequisite for strategic acting, which in return considers all options possible and defines a goal to achieve. In practice this rarely happens, and only a portion of the options available are considered. Some options seem intuitively more promising than others for human players (that this a hard problem for artificial players will be discussed in chapter 7 and section 11.5). One article discussing this aspect from a systems theory (the interdisciplinary study of systems in general) point of view was published by Niklas Luhmann (28) in 1969. There, humans take

decisions by the means of contextualisation (setting the decision in context with his own experience and knowledge) and reducing the complexity of a complex system through “Wertsetzung” (assigning of values). A human, says Luhmann, is incapable of making decisions “wertfrei” (without any values), otherwise it would be random selection. The paradigm of “considering all options” is therefore a pure theoretical construct.

But following the theoretical possibility of considering *all* options, a strategy requires information about all elements of a system that are relevant for a decision, and also a model of how these elements interact. Models don’t have to be purely virtual, but can utilise plans, maps, data charts, miniature models etc. which can be used to simulate actions; strategy games can provide the possibility space for such simulated actions. Acting within the strategic action space, following Nohr, resembles the concept of schemas in cognitive psychology: the models build from predetermined knowledge and subjective experiences work as schemas for decision making. The difference with games is that they not only serve as a space for simulated actions but instead also can generate experiences themselves, and the simulation itself becomes the center of attention.

Strategy and Emotions It also seems that the term of “strategy” was transformed throughout history. Speaking about strategic thinking already implies the existence of a certain rationality. Removing any emotional components from decision making is in fact an achievement of the long term process of affect control and self limitation. Elias names this skill in his study “Prozeß der Zivilisation” (29) (Process of Civilisation) the improvement of “Langsicht”. Langsicht is a composition of the German words *lang*, meaning “long” or “far”, and *Sicht* (view). A non-literal translation would be the English word “foresight”. Elias used this term for the description of behaviours of the European population, and described it as the skill of predicting the consequences of one’s own actions by evaluating the links within a causal chain. Antonio Damasio argued (30), that emotions play a critical role in decision making and can’t be separated. The lack of complete rationality however is important when designing parts of a strategy games other than its mechanics, i.e. the audio-visual context and its narrative play an important role in how decisions are made.

Strategy in Modern Mathematical Game Theory Another transformation of the word “strategy” took place in the middle of the 20th century when John von Neumann and Oskar Morgenstern published their book *Theory of Games and Economic Behaviour* (31) (and its preceding papers), what is credited today as the foundation

2. STRATEGY GAMES

of modern mathematical game theory. They use the term “utility” for a quantitative measurable variable that each player wants to gain in a game. Although players are supposed to act “rationally”, and maximise that variable, the authors observe that “there exists [...] no satisfactory treatment of the question of rational behaviour” and further: “The chief reason for this lies, no doubt, in the failure to develop and apply suitable mathematical methods to the problem; [...] the notion of rationality is not at all formulated in an unambiguous way. [...] the significant relationships are much more complicated than the popular and ‘philosophical’ use of the word ‘rational’ indicates” (ibid. p.9). If we consider several players in a game system, and that each player controls a certain set of variables, and the principles of how each player controls his variables are unknown to the other players, so is a qualitative “solution” of the maximisation problem impossible. Instead, each player develops a plan of how to react in the game in each possible situation. That plan is called a “strategy” (ibid. p.79). Neumann and Morgenstern do not cover any militaristic topics. Instead their framework is supposed to solve economic games with a clear distribution of information between players and finite predetermined rules. However, the models they describe are a method of reducing the complexity of a system and allow decision making in a finite action space. The strategies described by von Neumann and Morgenstern were a great influence on the *Wargames* of the 1950s. We will return to these games in section 2.1.

Governmentality and Simulations The equalisation of the two terms “strategy games” and “wargames” does not seem in step with the common definition of strategy games. A look into the genre “strategy” in shopping catalogues of any games reseller reveals titles like “Sim City” or “Railroad Tycoon”. A reason might be, that the term “strategy” went from its militaristic context through the process of universalisation and popularisation, making its way into almost every aspect of our daily lives: a *marketing strategy* or a *conflict avoidance strategy* is detached from a military context where troops are sent into battle. Instead, it refers to goal-oriented acting considering all options available (again: there is a discrepancy with the term “all” between theory and practice). Nohr (32) uses Foucault’s term “governmentality” to unify games that model militaristic, economic, or political processes. The term of “governmentality” is today associated with Michel Foucault who defined it as the “conduct of conduct” (33), what could be interpreted as the “art of governing”; with governing not being restricted to a political context, but also including a sociological or economical context, e.g. the term could refer to leadership of a state, or the leadership of a family. Both economic simulations and wargames require the player to constantly monitor the game context

and adjust variables to improve his position or prevent the loss of the game. This self-adjustment of the player within a control loop connects to Foucault's term of self-imposed repression; meaning that the player accepts (or tries to find) the values the game imposes as optimal and tries to adapt his actions to achieve those values. Nohr discusses this in regard to Link's theories of *normalism* (34) and uses *Sim City* as the main example. Link's *normalism* describes the mechanism of defining "what is normal" based on the statistical analysis of actual data of a population (as opposed to the normative; the pre-defined norm). Furthermore, subjects of a population - aware of the norm - now trend towards the "normal value" and in return affect the analysis. Ramón Reichert (35) also connects strategy games (here: government-games) to Foucault and discusses that these games have in common that, while they offer several degrees of freedom for players, they impose a particular *master narrative*. Knowledge is presented as scientific or canonical knowledge. Reichert discusses the *Civilization* series as an example, where "democracy is the most efficient form of ruling" (ibid. p. 199). It may also be noted that Reichert points out that government games normally have in common that the player is not visually represented within the game world. This is a detail which distinguishes strategy games from most other game genres.

2.1 The History of Strategy Games

The reenactment of war and military conflicts is not a phenomenon that came with digital media. Computer strategy games are the latest link in a chain of media transformations, starting with the earliest documented games in the late 18th century. It might be even argued further, that make-believe-games in which wooden sticks are treated as swords may have been played earlier; imitating real weapons of that time. Conversely, some game productions involve military personnel as consultants, or are (co-)financed by the military. Marcus Power showed in 2007 the increasing ties between the American Military and the digital games industry with the example of *America's Army* (36), a first person shooter developed through army funding with potential recruits as an intended audience.

Coming back to strategy games, a detailed overview of their history has been published by Sebastian Deterding in 2008. His paper "Wohnzimmerkriege" (37) (living room wars) outlines several significant epochs which we will briefly summarise in the following paragraphs.

2. STRATEGY GAMES

The early days (1780-1824) One of the first notable transformations from abstract games (such as Chess) into a more realistic medium is the “Kriegspiel” developed by the German mathematician Johan C. L. Hellwig and published in 1870 (refined in 1903) (38). That game is exemplary of several games published during that time. The game portrays an antagonistic militaristic situation where two players take the role of generals. Each general commands an army in the form of several game pieces. Each game piece represents a different type of unit (cavalry, infantry, artillery, etc.) and the game board consist of different wooden pieces that represent rivers, cities, armaments, etc. The game’s manual supplies the players with all necessary rules to move and attack enemy units. A more detailed description of the *Kriegspiel* can be found with Nohr (32). The intention of the game was to precisely model the mechanics of war, not to create an enjoyable game. Deterding (37) summarises the key elements as follows:

- *Modular terrain* The game board could be re-arranged, using different tiles such as plains, rivers, or cities.
- *Game Master* A person who acts as a bookkeeper and referee.
- *Non-Determinism* Some actions include non-deterministic elements which affect the outcome. These actions require a device for randomisation, e.g. dice.
- *Parallelism* Moving several game pieces per turn
- *Unit properties* As opposed to Chess, game pieces have properties such as health or moving points.
- *Spatiality* Tools to measure moving- and shooting ranges, e.g. rulers or markers
- *Limited intelligence* Both players and the game master have their own level of information, realised by three different tables in separate rooms.

Wargames become civilised (1880-1932) Both Nohr (32) and Deterding (37) compare Hellwig’s *Kriegspiel* to H.G. Wells’ game “Little Wars” from 1913, and point out that the latter is often seen as the birth of the “civil strategy game”. While Hellwig’s game has the clear intention to serve as a tool for the education of young cadets, Wells’ intention was to provide a form of social interaction (32, 37). Nohr also points out that this is already more aligned with Huzinga’s definition of play. Deterding iterates further over a few popular games from that time, and points out that (miniature) war gaming might have become successful due to the introduction of another phenomenon of that time: the tin soldier. These little miniature figures became affordable in the late 18th century due to industrialisation and mass production. And with the militarisation of the bourgeoisie they become a popular children’s toy. Otto Büsch wrote about the “social militarisation” in Prussian-Germany through conscription (39). A more recent

reflection of the original material by Büsch has been published by Wilson in 2000 (40).

Rebirth of a genre (1952) No significant development of the genre of strategy games during the second world war is known. But this, says Deterding, is probably more due to the lack of proper documentation of “trivial” items than the lack of people playing games. He defines the year 1952 as the next significant year in the history of strategy games: Charles S. Roberts created his own war board game, and published it with his company *Avalon Game Company* (later *Avalon Hill*) under the name “Tactics”. It set up two hypothetical countries, with typical post-World War II armies, engaging in war. The game was professionally produced and distributed through the *Stackpole Company* (which already had a reputation as a publisher of books on military affairs). It was the first modern commercial wargame as we know them (41). The game introduces, among other things, hex shaped tiles as a board layout which were imported from professional military simulations (42, 116). Also, tin solders were replaced with markers made of paper.

The Golden Age (1952-1980) After several successful publications of games by *Avalon Hill* (and others) followed what Deterding (37, 92) and Dunningham (41) call the “golden age”. They especially focus on the decade between 1970 and 1980 when several game publishers spawned, and the first strategy game convention took place: the *Origin*. The convention still exists today and had 10,030 and 11,332 visitors in 2009 resp. 2012 (43).

The digital exodus (1980-1985) On the peak of the popularity, in 1980, the main strategy games magazine *Strategy & Tactics* (founded in 1966) sold 37,000 copies, and the number of sold (paper¹) strategy games reached 2.2 Million. But in 1980 also the first computer strategy games were published. According to Deterding it is not possible to name “the” first strategy game, as the term was used incoherently. Some of the games published were board-/computer game hybrids, using the computer as the game master for a board game, or did not include a computer opponent. But it is safe to say that the introduction of 8-bit home computer sparked a boom of video games. Nearly every game idea was realised as an electronic version to see if it was feasible as an electronic medium. Naturally, with strategy games being one of the most popular genre of games, many of the published computer games were adaptations of existing game ideas. The transition also included themes: world war II and cold war games

¹Paper refers here to strategy board games where board and game pieces were cut out of paper.

2. STRATEGY GAMES

were the dominant setting.

Franchise worlds (1983-1987) In the late 1980s computer strategy games were influenced by another phenomenon, originating in the 1970s: fictional, but coherent and complete, worlds that were inter-medially connected. The success of Tolkien's *Lord of the Rings* (44) established a popular franchise, and many adaptations in different media were created. Tolkien's lore was combined with the game *Chainmail: Rules for Medieval Miniatures* and published as *Dungeons & Dragons* in 1974. *Dungeons & Dragons* is probably the most prominent so called "pen and paper" roleplaying game. It is a "make believe game" supported by a conceptual framework that defines which attributes objects and characters in the game world have, and how they may interact. Each participant plays a character who is commonly tracked on a sheet of paper, hence the name "pen and paper". The idea of a group of characters adventuring together resembles Tolkien's figure of "the fellowship". Similar examples can be found with *Star Trek*, *Star Wars*, etc. Two notable examples from the 1980s which Deterding presents are *Warhammer* and *Battletech*. Both systems were originally tabletop games, the modern variant of Hellwig's and Well's wargames. Each player has control over an army of (physical) miniature figures, and gameplay takes place in a model landscape. The first game system can be seen as a dark version of the fantasy world created by Tolkien, with all their orcs and elves. The latter models a universe where humans pilot huge battle robots in tactical combat. Both systems have in common, beside their popularity, that they focus significantly on the individual units. Most units used have their own abilities, lore, and the paper version of their description consists of multiple pages. Their re-mediation as computer games made these invisible and convenient to handle. It may be noted that *Warhammer* and *Battletech* are the only successful franchises that combined the medium of computer games with other game mediums. The next notable example is Blizzard's *World of Warcraft* more than twenty years later; again combining a computer game with board-, card-, and roleplaying games.

Simulations (1974-1990) Like Nohr (32), Deterding makes the observation that strategy games are hard to distinguish from simulation games. Both genres focus on algorithmic modelling (of complex systems). They also have a common ancestry in board games. In the 1970s several non-militaristic but strategic board games were published. A notable game from 1974 is *1829* which inspired Sid Meier to publish his popular computer game *Railroad Tycoon* in 1990. The creator of *1829*, Francis Thresham, also published the game *Civilization* in 1980 which in return was adapted

2.2 Strategy Games: The Delimitation of the Genre

as a computer game by Sid Meier as *Sid Meier's Civilization* in 1991. To avoid further legal disputes, Sid Meier's company *MicroProse* bought the publisher of the two board games *Hartland Trefoil* in November 1997. Ultimately, it should be added, that in 2001 another board game *Civilization* was published. This time as an adaptation of the computer game. One could argue, that this could be called "Re-Remediation", the transformation back to the original medium.

Trading card games (1993-2002) The last transition took place in 1993 and the following years, when the roleplaying games publisher *Wizard's of the Coast* published their trading card game *Magic: The Gathering*. The antagonistic situation of two players now took place in the form of two decks of different playing cards, each representing a resource or unit. The clou: not every copy of the game included the same cards. Players were supposed to buy "booster packs" that included new cards. It was impossible to peek inside the pack and it was unknown which cards a player would gain. This aspect, and the principle that more powerful cards were printed in fewer copies, created a trading scene. Powerful decks were created through the strategical collection and trading of cards.

2.2 Strategy Games: The Delimitation of the Genre

The brief recapitulation of the history of strategy games compiled by Sebastian Deterding (37) in section 2.1 ends with the question of why strategy games are so popular, regardless of their medium. His hypothesis is, that strategy can only exist in virtual space - the simulation - because there exists no natural situation where a person can have perfect information about a situation. It rather requires an interface to reduce the complexity of a situation to make it comprehensible and differentiable (ibid., p.108). It further requires a network of helpers and tools to create the visualisation of the situation and break down global strategic decisions to orders for the single individual. In other words, a game is the only space where perfect information and perfect control can exist to support strategic thinking and therefore strategy games are so appealing. Deterding also discusses the "remediation" of strategy games into computer games. In conjunction with McLuhan's dictum (45) that "the content of every medium is another medium", Deterding concludes that the first computer strategy games were simply strategy board games copied into computer games. The term "remediation" here refers to the process of developing into a new - independent - medium. Although the process of remediation does not seem relevant to our cause, the discourse cites a list of differences between

2. STRATEGY GAMES

computer- and board strategy games, published by Chris Chrawford (46) in 1981. This helps as another step for our definition of strategy games:

- *Network play* Communications Technology creates the possibility for two players to play with/against each other even when they are not physically at the same location.
- *Realtime* Given enough processing power, the computer can simulate the discrete steps of a game so quickly, that the game progresses in realtime in the perception of the human player.
- *Solitaire games* Implementing rules that make a computer program behave like a human player would, at least within the game space, creates the opportunity to play for a single individual alone.
- *Limited information* The program may choose to not reveal every detail about the gameworld to the player. Eliminating the necessity of auxiliary tools like different tables or blinds.
- *Cloaking the internals*. The mechanics and calculations, e.g. the dice rolls, are hidden in a layer of code; invisible to the player.

In this chapter we investigated several sources regarding strategy games, discussed their historical roots and media implications. Computer strategy games have their roots in the war-, board-, and role-playing-games that were developed in the 20th century. Besides the theoretical similarities, there are practical crossovers between wargames and economic simulations. In fact most wargames include simple economic simulations, most often the accumulation of resources to build more units to ultimately conquer the enemy. From these sources we elicited a working set of delimiting properties of digital strategy games for our work, which was first published in 2012 (47). The following properties of “strategy games” will delimit the games used in the following chapters:

- The base for strategic gameplay is a topographic map that defines relations between objects and space. Positions on the map can be either given in discrete (tile-based) or real values.
- A player does not incorporate an avatar to interact with the game world. Although some games use a unit/figure to represent the player on the map, the camera maintains a distant “bird’s eye” position.
- Two modi operandi of time are possible: turn-based or realtime. The latter includes simultaneous acting of all players, while turn-based makes this optional.

2.2 Strategy Games: The Delimitation of the Genre

- The player interacts with the game world through his assets. He can not directly manipulate the world but use game pieces he “owns” as a medium for his orders. It could be argued, that by clicking on a unit and then on a target the player actually manipulates the world, but we consider this rather “user interface interactions”. The target mouse click merely provokes the unit to shoot; the shot is not fired by a user avatar. For example, a target mouse click without a selected unit in range would have no effect. An example for direct manipulation by the player would be an omnipresent weapon attached to the player’s mouse cursor.
- Objects on the map may have properties. Objects are divided into classes that define the attributes and abilities.
- The interaction between objects is defined implicitly through their actions and abilities.
- The computer per se only acts as a bookkeeper (or gamemaster), making the mechanics of the game invisible. The game itself is a framework of rules. Computer controlled opponents are treated separately.
- The game requires at least two factions. Factions compete over the same or different goals. Each faction is controlled by a player. The definition of player here is transparent: it may be another local, network based, or artificial player.
- Each faction may have a different view on the game state. While there necessarily exists a well defined canonical state of the game at any time, the game rules may define what information may be revealed to a player. This feature is normally called *limited information* (as opposed to *full information*)

So ultimately the genre of strategy games might be better called “government games”, as the line between genres seems blurred. However, we will use a more restricting definition. This is simply for practical reasons to reduce the number of possible games modelled in our framework. To maintain a reasonable complexity of our experiments, we also introduced the following limitations to the term “strategy game”:

- Game mechanics can be divided into two categories, primary and secondary. The primary game mechanic is warfare. All actions that are immediately connected to destroying, hindering, or damaging objects are considered part of this.
- Secondary game mechanics such as economic or political processes act as support. Commonly economic mechanics are tailored towards unit production (i.e. resource gathering), and political processes evolve around diplomacy. Sometimes

2. STRATEGY GAMES

secondary mechanics can become crucial to the game play and help decide who wins.

Comparison between Sim City and Civilization To conclude this chapter, we would like to compare the two previously mentioned examples, *Civilization* (CIV) and *Sim City* (SC) in regard of the definition stated above:

1. Both CIV and SC feature a map which is segmented into tiles. Both games have different topologies: CIV maps are two dimensional, SC maps have different height levels (introduced in SC 2000).
2. In both games the player is not represented in the form of an avatar. Some CIV games represent the player's keep in form of a throne room in the nation's capital, but it is never located on the map directly. SC has a similar mechanism with the difference that the mayor's mansion can be placed on the map directly. However, both game series do not feature a player's avatar.
3. CIV games are turn-based (multiplayer plays a special role with simultaneous turns though), while SC games progress in realtime
4. In CIV players have to use units to change the world. The best example are the player's worker units which are needed to create various types of tile improvements, e.g. roads or farms. In SC the player can directly manipulate the world: either using terraforming tools and placing structures directly with his mouse cursor.
5. Both games feature maps with objects. These objects are grouped into classes by attributes and abilities CIV features units and cities, objects in SC are structures grouped by function.
6. Each unit type in CIV possesses several abilities, e.g. to move around, build, or shoot. In SC buildings have, with a few exceptions, no abilities they can invoke. Instead the game features a macro model of several potential fields. For example, crime is not presented as agents committing actual crimes, but instead police stations influence the potential field of average crime. This field can be directly visualised in the game's main view. However, buildings do not direct act or interact.
7. Both game series hide the actual calculations of the next game state: for example, when two units engage in combat the result is calculated and just presented as

2.2 Strategy Games: The Delimitation of the Genre

an animation to the player. In SC the complete computational model is hidden from the player, and only represented as a few indicators; either directly in the game world (houses displaying another graphical state when being abandoned), or in the form of diagrams.

8. CIV features a variable number of factions, SC has no concept of factions or teams. Therefore the point of a different perspective does apply to CIV, but not to SC.
9. Warfare is not a part of the SC series. It is however the primary mechanic of the CIV series.
10. Economics are the primary mechanic of SC, but only play an ancillary role in CIV.

Although both game series have similarities, we consider only CIV a game which suits our definition. Sim City falls outside of our categorisation because of points 4, 6, and 8. The main separation however is based on points 9 and 10.

2. STRATEGY GAMES

Chapter 3

Related Theoretical Frameworks

Our work puts the object of investigation - strategy games - into the context of computational intelligence in games research. Our main objective is to create strategy games and evaluate them in terms of how interesting and appealing they are to human players. It therefore seems relevant to introduce what definitions of “interesting” or “entertaining” in terms of games have been published. In the following, we will present what research has been published in regards of gameplay experience: the idea of an interesting game, i.e. what makes a game enjoyable. The research presented herein consists of conceptual frameworks and related method of measurements. The chapter is divided into quantitative and qualitative methods though some methods could be categorised as both.

Chapter 4 will then present related work that is connected to the aspect of content generation. Not all of the methods presented here directly relate to our work on strategy games. However, we discuss them to set our work in context with other approaches. Especially the work regarding the emotional state of the player is not applied in this thesis, but we see this as relevant if one would add the aspect of adaptivity to the SGDL framework. We consider it future work to consider it players’ preferences in the generation of strategy game mechanics.

3.1 Modelling Player Experience: Qualitative Methods

The term “fun” is highly debated within the academic community, and consensus seems to be that is a rather ambiguous concept and is oversimplifying the experience or motivation to play a game. The reason why somebody plays a game, and which game he plays is very diverse. This section tries to approach the question *why* and *how* people

3. RELATED THEORETICAL FRAMEWORKS

play games, and what theories can be used to identify answers to both questions from an academic angle.

We start with the research done by Lazzaro et al. They argue that people play games to change or structure their internal experience (48) and feel emotions that are unrelated to their daily life. While we will discuss the definitions of emotions further in section 3.1.5, Lazzaro et al. identified four key features that may unlock emotions in players while interacting with a game. They point out further, that the requirements for entertainment software differ in some key aspects from software for production purposes (e.g. office suites or CAD). Lazzaro therefore propose the distinction between *User experience* and *Player experience*. While the former also partially applies to games (e.g. a game shouldn't contain bugs or present confusing input methods), the latter does not apply to production software; although gamification in serious software is another uprending topic.

The studies conducted by Lazzaro et al. indicate that people prefer titles which provide at least 3 out of 4 features. They have identified about 30 emotions that are connected to four quadrants, whereas different emotions arise from different mechanics and playstyles. Moreover, interesting games let the player move in between these quadrants to create a diverse experience:

- **Hard fun** meaning challenge and overcoming problems in the game. This usually generates either frustration or fiero (Italian for "personal triumph").
- **Easy fun** refers to the enjoyment of intrigue and curiosity. Absorption may lead to wonder, awe and mystery. The absence of easy fun will make the player disengage with the game very easily because it is too frustrating.
- **Serious fun** (previously named *Altered states*). Players reported as a motivation that games could change "how they feel inside". Meaning that by clearing a level they could clear their mind, or that a game made them avoid boredom. The lack of serious fun usually makes a game feel like "a waste of time".
- **The people factor** this summarises the social interactions players have in a multiplayer game. Coming from social experiences like cooperation or competition, players would report emotions like amusement, schadenfreude or nache. The people factor can also motivate people to play who normally state that they are "non-players", also it motivates people to play games they don't like.

Malone (49) instead proposes three categories: *challenge*, *fantasy*, and *curiosity*. For "challenge" he mainly refers to goals and that the simpler the game is, the more obvious the goal should be. On the other hand, the more complex a game is, the better structured the goal should be. Players should also be able to tell if they are getting closer

to the goal or not. He further categorises *uncertain outcome* and *hidden information* under challenge; points that are also picked up by Juul (50). For “fantasy” Malone divides between *extrinsic* and *intrinsic* fantasy. Extrinsic fantasy could be seen as an early idea of “gamification” (applying game principles to non-game context), where a game is laid on top of an existing curriculum. The abstract game is just a metaphor for a conflict or value in the original system, e.g. children must answer basic math questions to advance on a “Snakes & Ladders” (51) game board. Intrinsic fantasy on the other hand refers to the concept that the game task also depends on the fantasy as a setting. Malone gives as an example the simulation of running a lemonade stand: the game task (calculating the right price, i.e. making mathematical decisions) is embedded in the system. Malone adds that intrinsic fantasies are more interesting, as they are generally less abstract. For the last category “curiosity”, Malone distinguishes between *sensory*- and *cognitive*-curiosity. Sensory curiosity refers to the concept of changing patterns in light and sound that create different stimuli. Cognitive curiosity seems inspired by Shannon’s concept of information (52): at the beginning of the game, the player’s information about the game’s world and mechanics are incomplete. One motivation to play the game is to overcome that inconsistency.

Measuring how much a player enjoys a game is more elusive than measuring more traditional performance metrics, such as time on a task or number of errors, which have been successfully applied to productivity applications (53). A measuring framework has to solve the issue of *comparability*, i.e. how to compare the “fun” of killing a horde of virtual aliens in a first-person-shooter to the experience of playing a puzzle game (e.g. SpaceChem(54)). Furthermore, it has been tested that the idea of “fun” differs between different playing styles (55), and even more generally between men and women. Also the cultural aspect plays a role (56). So far a canonical and clear taxonomy and vocabulary to describe a gaming experience has not been finalised, and it is debatable if this is actually possible. We will try to list some approaches that have been made to define and articulate the general experience with but not limited to computer games.

3.1.1 Self-reported data

A widely used approach to capture the experience with a game (or a computer program in general) is to let players express their own thoughts and preferences about the experience while or after playing a game. Viable methods in game experience evaluation are (among others) questionnaires or focus groups, i.e. supervised playing. There have been a variety of questionnaires proposed for different aspects of the gaming experience. A well-established survey method is the “game experience questionnaire” (GEQ). It has

3. RELATED THEORETICAL FRAMEWORKS

been tested to reliably measure the seven dimensions sensory and imaginative immersion, tension, competence, flow, negative affect, positive affect, and challenge (57).

An important point has been raised by Yannakakis and Hallam in 2011: even though self-reported data offers a direct approach to affect detection, they are prone to self-deception, intrusiveness and subjectiveness. The used scheme has a significant impact on the validity of the reported data. In their paper, Yannakakis and Hallam compare two popular schemes for self-reported data: *rating* (or *scaling*) and *preferences*. In both schemes participants are asked a range of specific questions regarding the experience with or of something. The majority of psychometric user studies using the method of rating, where participants are asked to rate their preferences on a scale, the commonly used Likert Scale (58). They point out that when commonly averaging data over all participants the subjectivity of the rating is eliminated. Furthermore, the ordering of the possible answer and the internal cognitive processes, cultural background, temperament, and interests is relevant. Yannakakis and Hallam compare the use of scales in self-reports to *preference modelling* (59), where participants are asked to express pairwise preferences, e.g. “I prefer A over B”. This would eliminate inconsistencies like rating two features as high that contradict each other. Yannakakis and Hallam present a variety of user case studies and a statistical analytic comparison of those. They conclude that rating questionnaires appear more susceptible to order-of-play effects than preference questionnaires. Furthermore, they found it interesting that reported scales and preferences did not match exactly, suspecting that the order-of-play is the most significant factor.

3.1.2 Requirements elicitation

Computer games research in general is a fairly young field compared to other disciplines of computer science, including software engineering. It therefore seems beneficial to look into other fields, where extensive research on usability and effectiveness of software has already been conducted. In a games context “effectiveness” could be translated as “enjoyability” if we declare that the purpose of a game is to entertain the player. We think that playtesting games can be compared with user testing non-entertainment software. Methods that have been established through and from the software engineering community should therefore apply. Naturally, artistic elements such as game art, sound, or narratives play a minor role in non-entertainment software. However, methods such as “requirements elicitation” should apply. Requirements elicitation describes a number of methods to gather the information about the required features of a system from end users, customers, or stakeholders. The task is non-trivial as one can never be sure if all

3.1 Modelling Player Experience: Qualitative Methods

the required information is compiled, and the term “elicitation” is commonly used in literature (as opposed to “gathering”), because simply asking the customer would not be sufficient. In the context of games an argument would be, that designing a good game requires background knowledge in game design, psychology, or narration – to just name a few – besides the actual technical skills to realise it. The open question “What would be a fun game?” will probably produce very broad answers.

The paper by Nuseibeh and Easterbrook (60) from 2000 presents several aspects for the creation of software requirements. The term “Requirements” there refers to technical requirements of software, e.g. data modelling, behavioural modelling, or domain modelling. The authors present a survey about currently used or published methods, resulting in a “roadmap” and an overview about the field of Requirements Engineering in the year 2000. This roadmap is later picked up up by Cheng and Atlee (61) in 2007, who present an updated taxonomy of papers published in the field. One of the conclusions drawn in that paper is that the Requirements Engineering community would greatly benefit from industrial organisations providing data of “real world” projects to the community. This problem exists in the computational intelligence games community as well. Quantitative analysis of large scale industry-strength data such as player metrics has been proven to be a valuable tool in player modelling or strategy extraction.

Ethnographic studies Ethnography is a technique coming from social and cultural sciences to explore the life, ways, and values of a cultural group. An ethnographic study tries to describe the point of view of that group, as objective and independent from the researcher as possible. The idea is that the researcher imposes no bias on the data he collected. In practice however, this can only be guaranteed to a certain extent. Although more a technique than a specific field, the term “ethnographic research” has become associated in the 20th century with cultural anthropology, when western researchers travelled into non-western cultures. Among others, two publications are often credited as the basis of the field: *Argonauts Of The Western Pacific* (62) or *Deep Play: Notes on the Balinese Cockfight* (63), both describing rather exotic tales from a western point of view of that time. Ethnographic studies in the field of software engineering research are normally concerned with the software creation process, i.e. how teams work, what resources are need, which external factors may hinder the process etc. The paper by Lethbridge et al. from 2005 (64) presents a good overview of techniques used by ethnographers. The authors present a taxonomy divided in three “degrees” (or categories), each one requiring a different grade of involvement of the persons studied.

3. RELATED THEORETICAL FRAMEWORKS

The three grades also differ (according to Lethbridge et al.) in the three dimensions: reliability, flexibility, and the use of resources. Discussing the listed techniques in detail here in regard of the game production process seems rather uninteresting, as the differences are probably too few. Instead, ethnographic studies in games research focus on the quality of the end product, or – to be more precise – the users, i.e. how do people play games? Communities of online role-playing games such as *World of Warcraft* are an often used research environment, studying behaviours of players and their interaction with the game and with each other (65). Other studies tried to compare the players of eSport tournaments with traditional professional sports (66). Ethnographic studies in the field of computational intelligence in games are unknown to the authors of this paper, but since agent design is an important discipline in our field, it seems interesting to create an agent for a specific simulation game after observing the actual environment. In an article by Robinson et al. from 2007 the authors try to generalise a number of observations and challenges the authors made and faced with studies - focussed on software engineering - done in the previous decade. Especially the relationship between the researcher and the subject being researched seemed to be a challenge. Speaking about one of their studies, the authors state: “In the object technology study our experience was slightly different. Here, our *participants* were not people but contemporaneous artefacts. However some of the artefacts represented events or movements that we had lived through ourselves, e.g. notices for a workshop or conference that one or other of us had attended, a journal paper describing the relationship between expert systems and object-based systems which one or other of us had read with a different purpose many years ago”. This exemplifies the problem any ethnographers face: as the common ethnographic view is that the researcher comes from a different culture than the culture being point of the study, it is easy for him to consider everything as “strange” and preserve an objective point of view. On the other hand, a culture or field that is highly specialised such as software engineering requires more background knowledge than an informal theoretical framework to understand “what is going on”. Transferred to games this relates to game researchers being gamers themselves; or game makers making the games which they want to play, often limiting the target audience of their game (unconsciously) to a “core audience”. The term (hard)core here stands in direct opposite of a more recent stream in game making: causal games for a broader audience (67). To our knowledge, this effect has never been researched. On the contrary some game companies use this advertising slogan: “By gamers, for gamers”¹.

¹Slogan of Interplay Entertainment (<http://www.interplay.com>)

3.1.3 Personality

Lindley and Seenersten propose a framework (68) based on *attention theory* which models interaction with a game as *schemas*. A schema is understood as a cognitive structure that distributes cognitive resources (such as attention) to motor outputs of game play in response to the ongoing perception of an unfolding game (game play pattern), input pattern, or story pattern etc. The framework further specifies that schemas are organised in a hierarchy, mirroring the structure of goals and tasks set for a player. The high level goal might be finishing the game, what might be decomposed into finishing each level, which requires the task of killing each monster in the level and so on. Lindley and Seenersten hypothesise that schemas are a fundamental structure for the motivation to play. Their method to recognise patterns or schemas seems to be a pure manual task: recording and analysing logging keystrokes, mouse movements, gaze tracking, and think-aloud protocols under and in controlled test environments/levels. The framework was initially proposed to detect the correlations between designed features and achievement emotions/behaviours of players, but Lindley and Seenersten further propose that schemas could be used to identify players' personalities and preferences. They argue that the automatic detection of schemas and schema execution may indicate which form of satisfaction particular players are seeking (69). These experiments are mainly described in (70). Together with Nacke, Lindley and Seenersten describe the experiments to measure emotional response to certain game play elements. They examined the correlations between external recorded data, e.g. physical measurements (such as described in section 3.2.2) and through questionnaires about experiences. The results indicate a correlation between emotional responses and game play features, especially visual attention patterns.

3.1.4 Flow

According to a theory in the field of psychology, *flow* is a mental state where a person is fully immersed in a task and its success. It is assumed, that this state is reached when the limbic system, controlling a person's emotions, and the neocortex, the part of the brain that is linked to consciousness and cognition are fully synced. Physically, this state of adaptation of the mind and a person's environment can be quantified and verified by a measurement of the heart rate variability. The theory of "flow" has been developed by Mihály Csíkszentmihályi (71) in 1991, starting the research in the 1960s. The psychological theory has several main prerequisites for a person to reach the state of flow with a task when:

3. RELATED THEORETICAL FRAMEWORKS

1. The task has clearly defined goals, and
 - (a) The task provides immediate response about the person's performance
 - (b) The task may be autotelic, i.e. the task may be to perform the task
2. The requirements of the task matches the person's skills and abilities. The task must not be too overburdening nor underchallenging (as illustrated in figure 3.1)

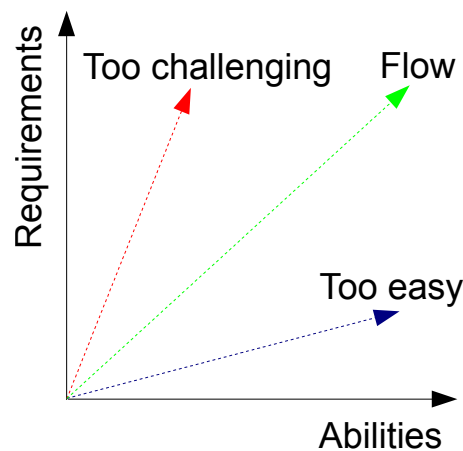


Figure 3.1: Illustration of the concept of flow, the relation between a task's requirements and the abilities of its performer.

The experience of flow is characterised as:

- having complete control over a task
- feeling a task becoming effortless
- losing the sense of time

The concept of “flow”, how to achieve it, and what its benefits are have been studied in regards of tasks in other fields before, e.g. online consumer behaviour (72) or Human-Computer-Interaction (73). It has been applied to games by Sweetser and Wyeth in 2005 (74), which features a comparative study of two games in regards of supporting the requirements for a flow experience by the player. Game experts were asked to rate the games *Warcraft III* and *Lords of EverQuest* in regard of several statements such as “games should increase player skills at an appropriate pace as players progress through the game” on a numeric scale between zero and five. The authors concluded that some criteria apply more to other genres of games than the two (strategy) games which were part of the experiment. For example, the sense of having control was identified as being more important for roleplaying games, or being immersed applied more to first person shooters than strategy games. The authors also identified shortcomings

of the methodology, stating that some criteria could not be efficiently rated through game experts but require actual playtesting by players with different skill levels and familiarity with the genre, e.g. learning curves are difficult to describe retrospectively by people who already mastered a game. The initial study was recently verified by Sweetser et al. in a new publication (75), reporting that most of the original findings were confirmed with new expanded heuristics. An overlap between immersion and concentration was clarified in the new paper, as the boundaries in the original study were rather blurry. Overall, Sweetser et al. propose a detailed heuristic for measuring “GameFlow” in real-time strategy games, taking into account typical features such as races, campaigns and missions, or gameplay concepts such as “upgrading units”.

Koster (76) also discusses the effect of flow, but states that flow doesn’t necessarily require a cognitive understanding of the process by the player (ibid. p. 98). Instead flow is created through successful pattern recognition. Pattern recognition could mean deciphering of the correct sequence of actions to jump through a certain part of a level, the correct combination of puzzle pieces, or the right order of moves in a strategy game. For every new pattern the brain recognises, it releases a jolt - so Koster says. If the flow of new patterns slows, then the player would feel boredom. Koster also argues, that flow doesn’t equal fun, as “flow” is not a prerequisite of fun. Vice versa, Koster argues that there are many activities which create “flow” but are not necessarily seen as fun, e.g. meditation. Instead, says Koster, games are fun as long as the player can learn something (new patterns) from it.

3.1.5 Emotional State

Picard et al.(77) proposed five dimensions of emotions to model the affective state of a student. The model is used to validate and control the actions of a companion agent for learning. They start with discussing several frameworks of emotions from psychology which propose several different numbers of possible emotions (78, 79, 80). Picard et al. build a five-dimensional model, that they cross with the concept of (de-)constructive learning, resulting in a two-dimensional model. This model describes the emotional state of the learner and its positive/negative effect on the learning process (similar to Csikszentmihalyi’s Flow theory (section 3.1.4).

In their further work Picard et al. differentiate that learners have needs which are different from other users and that how the user feels is relevant (81). While this at first sounds trivial, it is imperative that designers (of any human-computer interaction) should consider that the user’s emotions not only contribute to irrational behaviour but also play a role in rational decision making (82). Picard et al. back this up by a

3. RELATED THEORETICAL FRAMEWORKS

short survey of studies about the impact of negative emotions. the authors continue with the assumption that interactions which satisfy “experimental emotional needs” are exclusive to human-human interactions, but could also be fulfilled by non-humans (i.e. pets or computer systems). Their example continues with an interrogation system for patients. There, patients gave a positive response about the computer system because they felt that it lowered the barrier between the doctor’s status and their own “working class” status. It was given that the computer was less “judgemental”. It seems that their regular doctors may be more condescending, and that patients felt less uncomfortable with a neutral computer. Following the paper by Picard and Klein, there are two issues mentioned that may also be relevant to game designers: (1) that experts may see through the user interface and realise the underlying system, therefore the system may fail to satisfy their needs (i.e. a game where mechanics are too obvious may become unappealing to a skilled player) and (2) the issue of privacy where a user is not aware of the adaptation capabilities of the system and therefore shares more details. In terms of games this might be a lesser issue, as adaptive systems are usually marketed as a unique selling point (e.g. the AI director in *Left 4 dead*¹).

Picard and Klein continue by discussing the issue of systems which imitate human behaviour but fail on the subtleties (also known as the *uncanny valley* by Masahiro Mori (83)). This problem can ruin the interaction and therefore “building machines that are as much like people as possible” may not be the primary goal, and people should have control over the sensing process, as people tend to develop a defensive stance when they feel manipulated. Following Picard and Klein, it may also be concluded that different personalities may prefer different responses from a machine (84).

Similar work was done by Ravaja et al. (85) where the correlation of different kinds of games (e.g. the view from which the game is played, naturalness of the game, amount of violence) and self-reported emotions were examined. They conclude, that to compare the “goodness” of different games, it is not sufficient to compare them on a basis of one single emotion, but one should instead look at emotional patterns (profiles) associated games. Furthermore, a game does not generally have to generate positive emotions to be “good”. Thrilling or horror games ideally generate the emotions of fear and anger to be rated as “good”, yet these emotions are generally not categorised as positive emotions. Ravaja et al. conclude that a game at best stimulates a strong emotional reaction.

¹<http://www.l4d.com>

3.1.6 Immersion

Immersion related to games is often used as a synonym for involvement or engagement with a game. Sweetser and Wyeth (74) put immersion on the same level as the flow experience: “deep but effortless involvement”, which makes it difficult to distinguish from other concepts. Ermi and Mäyrä (86) propose a three layer model: sensory-, challenge-based-, and imaginative-immersion. *Sensory* refers to the perceptual impact on the user, *challenge-based* refer to the cognitive and motor skills of the user, and *imaginative* refers to the immersion within the virtual world, the richness of the narrative structure of the game.

Brown and Cairns (87) show that players, when talking about immersion, refer to the involvement within the game. Furthermore, Brown and Cairns describe a model of three stages of immersion: engagement, engrossment and total immersion (or presence). Each stage also seemed to be only achievable after a certain time: *Engagement* here refers to the simple fact that the player is willing to spend time with the game, *Engrossment*, the second level, raises the game to an experience that affects the emotional state of the player. The third level, *total-immersion* was described as a “cut-off” from reality in all matters.

Calleja (88) proposes the term “incorporation” instead of immersion since the first has been widely used with conflicting meanings, diminishing its analytical value (ibid. p. 4). Incorporation however, says Calleja, is the “internalization of spatial and other frames of involvement“. Overall, he discusses six aspects of involvement.

1. **Tactical involvement** decision making, planning, opportunity costs, the pleasure of decoding
2. **Performative involvement** movement, plan execution, “button pressing”
3. **Affective Involvement** cognitive, emotional and kinaesthetic feedback loop, emotional arousal, cognitive challenge, optimal level of arousal
4. **Shared involvement** interaction with the world and other agents, human agents allow a wider range of communication
5. **Narrative Involvement**: designed narrative, personal (i.e. emerging) narrative, “The lived experience is stored into players’ memory”
6. **Spatial involvement** locating one’s self within a wider game area, internalize immediate location

3. RELATED THEORETICAL FRAMEWORKS

Calleja picks up the aspect of spatial immersion especially in regards of strategy games (ibid. p. 140):

[players can] get absorbed in the cognitive challenge of a strategy game or stroll leisurely in aesthetically appealing landscapes. [...] the appeal of beautifully rendered environments can be particularly powerful when contrasted with unattractive everyday surroundings

Like many other game types, strategy games can be played both competitively and casually. However, not all released titles support both modes.

3.2 Modelling Player Experience: Quantitative Methods

The previous section presented models and theoretical frameworks that have been proposed to model player experience in a more explorative way. This section presents data collection and survey methods, that have been used or proposed in experiments or a professional context to measure the effects of games on humans while they are playing. Some methods originate from other fields, others are only applicable to games and are therefore considered as “player experience measurements”. This section presents both methods that use player centric measurements and methods which rely on in-game data.

3.2.1 Interest

The question of whatever a game is “interesting” is a very broad question. When asked, players may refer to the graphics or the narrative structure. Yannakakis and Hallam proposed (89) that an “interesting game” may rather refer to behaviour of non-player character agents in the game. They formalised three components that make a predator/prey-game interesting and showed in an experiment that their definition was very robust against different playing styles in a limited context. Yannakakis and Hallam used a modified version of the classic arcade game *Pac Man*. The general game mechanic remained the same: the Pac Man agent has to “eat” all the pills within a maze while avoiding the ghosts who are trying to chase him. If Pac Man eats a “Power Pill”, the ghosts become edible for a short time and the Predator-Prey roles become reversed. But instead of focussing on Pac Man, i.e. the player, Yannakakis and Hallam put the ghosts and their strategy to chase Pac Man in focus. The Pac Man agent was controlled by a variety of static strategies, but the ghost time was controlled by a controller consisting of a neural network which evolved through neuro-evolution. While they also evolved

3.2 Modelling Player Experience: Quantitative Methods

controllers for optimal behaviour, in terms of efficiently chasing the Pac Man agent, did they focused on evolving controllers which would provide “interesting” behaviour. The following three components were used to evaluate a controller’s performance:

- Challenge (T). Meaning that escaping the predators may not be too hard or too easy, expressed in the difference between the average and maximum times steps necessary to kill the prey:

$$T = [1 - (E\{t_k\}/\max\{t_k\})]^{p_1} \quad (3.1)$$

where $E\{t_k\}$ is the average number needed to kill the prey over N games; $\max\{t_k\}$ is the maximum number of steps required over N games; and p_1 is a weighting parameter.

- Diversity (S). The strategies of the predators to kill the prey should vary. This can be expressed as

$$S = (s^2/s_{\max}^2)^{p_2} \quad (3.2)$$

where s^2 is the sample variance of t_k over N games; s_{\max}^2 is the maximum value of s^2 over N games. p_2 is a weighting factor.

- Aggressiveness (H). Predators should use the whole game space instead of just following the prey. This can be formalized as the entropy of the cell visits (given that the game has a discrete space):

$$H = - \sum_i \frac{v_i}{V} \log \left(\frac{v_i}{V} \right) \quad (3.3)$$

where V is the total number of visits of all visited cells (i.e. $V = \sum_i v_i$) and p_3 is a weighting factor. This can be normalized to $[0,1]$ with:

$$H_n = (H/\log V)_3^p \quad (3.4)$$

All three components were then combined into a linear combination:

$$I = \frac{\gamma T + \delta S + \epsilon E\{H_n\}}{\gamma + \delta + \epsilon} \quad (3.5)$$

where I is the interest value of the game and γ, δ, ϵ are weight parameters.

Although only a varied version of Pac Man with different playing styles was used, Yannakakis and Hallam propose that this function could be used for any predator/prey game following their definition. Although the calculations are specific to Prey-Pradator

3. RELATED THEORETICAL FRAMEWORKS

games, the characteristics Challenge, Diversity, and Aggressiveness measures could be applied to strategy game bots as well. It is debatable if the definition is sufficient to categorise the whole (strategy) game as “interesting”. Compared to the games used by Yannakakis and Hallam, strategy games include more and more complex aspects which contribute to the player experience. Generally, the factors “Challenge”, “Diversity”, and “Aggressiveness” seem to be applicable, but in a different context. Challenge in strategy games does not rise from spatial navigation or reaction time. Instead, a strategy game gains challenge from its rule complexity or other aspects. Diversity, the number of possible strategies to win a game, is a highly relevant aspect to strategy games. This normally creates significant value to replay a game. Aggressiveness seems less relevant, and would need a further definition. There are a variety of roles and agents taking part in a strategy game so that this seems inapplicable on such a high level.

3.2.2 Physiological measures

Asking players about their emotions and their experience with a game has the problem that participants, knowing that their answer is recorded, will sometimes give the answer they think the investigator would like to hear. This can also happen without the participant even realising it. Ratings may be affected by the participants cognition than resembling what is actually occurring (90). Instead, following the work that has been done in the Human Factors research community (91), the usage of physiological signals as an approximator of fun has gained some interest.

- **Galvanic skin response (GSR)** GSR measures the conductivity of the skin. It can be best measured at specific sweat glands (the eccrine sweat glands) using two electrodes. Those sweat glands, located in the palms of the hand or the sole of the feet, change the conductance of the skin without the necessity that the participant is actually sweating, as the sweat might not reach the surface. GSR is suspected to be linearly correlated to arousal and cognitive activity (92).
- **Cardiovascular Measures** These measures include blood pressure, heart rate, heart rate variability and signals gathered by an electrocardiogram (EKG).
- **Respiratory Measures.** This refers to the breathing frequency and the volume of air exchanged through the lungs (depth of breath). Emotional arousal directly affects these measures.
- **Electromyography** Refers to the electronic activity that occurs when muscle are contracted. Attached to the face and jaw, these can be used to recognize movement of “frown muscles” (over the brow) and “smile muscles” (over the

cheek).

Mandryk et al. (93) showed that physiological signals can be used as indicators for the evaluation of co-located, collaborative play, and that the results will correlate with the reported experience of the player. Tognetti et al. also found that certain physiological measures (e.g. GSR) correlate with the reported player preference, while other (Heart rate or temperature) don't. Although physiological features can be seen as an objective measurement and are (for most computer games) not task dependent, as the energy to play a game doesn't really depend on the game being played, should qualitative measurements such as *curiosity* and *flow* should rather be taken from the analysis of the data gained of the player interaction from within the game (player metrics).

Yannakakis and Hallam (94) used the physical-activity game *Bug Smasher* to research if physiological measures correlate with preferences over game-variants. Moreover they used preference learning (neuroevolution) to construct entertainment models and feature selection to isolate the significant signals.

While this is an established technique to gather player feedback, it would be interesting to see if this actually applies to a player playing strategy games as they are not a physical activity, and seldom contain the element of horror or suspense. In fact, no strategy game with these characteristics is known to the authors. Nevertheless, strategy games certainly evoke emotions: succeeding in executing a certain strategy may trigger the emotions of joy and triumph. It is questionable though if these emotions occur in a sufficient frequency compared to e.g. a first person shooter.

3.2.3 Tension in Board Games

In 2006 Iida et al. proposed a fitness measurement based on outcome uncertainty. They used the game *Synchronized Hex* as a case study. They approximate the chance of winning for each player through a number of self-play rollouts similar to a Monte Carlo Tree Search. We will use the approach presented here in our fitness presented in section 8.2.

Once the probability distribution $P = (p_1, p_2, \dots, p_n)$ of the chances of winning for n players is obtained, we can use the standard information theory formula (52) for outcome uncertainty (entropy)

$$H(P) = - \sum_i^n p_i \log(p_i) \quad (3.6)$$

with n being the total number of possible outcomes and p_i the probability of outcome

3. RELATED THEORETICAL FRAMEWORKS

i. Here, this means n being the number of players (plus potentially the possibility of a draw) and p_i the probability of player i winning the game resp. the game ending in a draw.

Analogously, the outcome function for a given position G in the game can be formulated as an entropy function:

$$U(G) = - \sum_i^k p_i \log(p_i) \quad (3.7)$$

where k is the number of possible outcomes (player one wins, player 2 wins, draw etc.), and p_i is the probability that the outcome of the game will be i .

It is unknown to the authors if this concept was ever formalised into a fitness functions for games. We present our own interpretation in section 8.2.

3.2.4 Measures from Combinatorial Games

In 2008 Browne (95) published extensive work on fitness functions for combinatorial board games for two players, e.g. Chess, Checkers or Go. Explicitly noteworthy here is that these measurements use automatic playthrough, like most of the fitness functions used within the SGDL framework (will be presented in chapter 8). Artificial agents play games expressed in Browne’s *Ludi* language (discussed in section 4.4.4) while different indicators are tracked. Browne created aesthetic measures based on Birkhoff’s theorem (96)

$$M = f\left(\frac{O}{C}\right) \quad (3.8)$$

whereas the aesthetic measure of an object M depends on the order O (simplicity, symmetry etc.) and its complexity C (number of components, level of detail etc.). Browne’s proposed measurements in a brief overview are:

- **Complexity** Taken from Pell’s METAGAME generator (97), complexity refers to number of move options for each piece on the board plus the number of winning conditions.
- **Movement Type** Accounts for the existence of different movement types in the game out of a list of possible types, observed from several existing games (e.g. add a piece, remove, capture etc.)
- **Goal Type** Similar to Movement Types, a game could have one or various goals out of a list of goal types observed in existing games.
- **Branching Factor** Refers to the number of move choices per turn for a particular player.

3.2 Modelling Player Experience: Quantitative Methods

- **Response Time** Time it takes a computer agent to formulate a move. It is assumed that if it takes an agent longer to calculate the best move, it will also be harder for a human player.
- **Move Effort** The number of moves a player has to plan ahead to make an adequate move. It is proportional to the information a player must process each turn.
- **Search Penetration** Indicates if the game tree of lookahead moves for a player is rather deep and narrow or rather shallow and wide.
- **Clarity (Variance)** This indicates how clear for a human player it is to differentiate between promising and unpromising moves. In terms of a MCTS (see Monte-Carlo Tree Search in section 4.3.1.2) agent this is the variance of the evolutions of the moves available.
- **Clarity (Narrowness)** Like the various measurement, this relates to the game's clarity. It determines if normally a small clear choices stand out as the promising moves each turn.
- **Convergence** Approximates the trend of the number of possible moves available to a player rather to de- or increase throughout the game.
- **Uncertainty** Refers to the same concept as Iida's work (as in section 3.2.3) but instead of rollouts of self-play Browne used a custom function that takes the lead graph of the players into account.
- **Drama (Average / Maximum)** The number of moves the eventual winner spends in a trailing position
- **Permanence** Indicator if a player can immediately reverse or recover from a negative effect that his opponent just played.
- **Lead Change** Number of times the leading player changes throughout the game. We will use this measure in chapter 8.3.
- **Stability** The fluctuations of players' evaluation (scores) throughout the game.
- **Coolness** Indicates if players are forced to make a move that is not beneficial to them.
- **Decisive Threshold** The point in the game where it is nearly clear which player will win this game. The term "threshold" refers to a reasonable degree of confidence.
- **Decisiveness** The measurement of how fast a game will end once the decisive threshold is reached.
- **Depth (Discrepancy)** The variance of how many moves a player has to plan ahead.

3. RELATED THEORETICAL FRAMEWORKS

- **Board Coverage** Considering board games, this measurement indicates if the action normally takes place on certain parts of the game board (e.g. middle) or is widely spread over the whole board.
- **Momentum** Tendency of players to defend a leading position with their consequent moves.
- **Correction** Tendency of a player's score, that has gained the lead, to continue downwards (sometimes referred to as "rubber-band effect" (98)).
- **Control** Degree of which the leading player can limit the moves of the trailing player.
- **Foulup factor** The chance of a player making a serious mistake by overlooking a move.
- **Puzzle Quality** An estimator of difficulty for a puzzle. It is measured on the counter-intuitiveness of puzzle, i.e. a puzzle that requires a first move that is different to what 999 of 1000 players propose (99).
Browne further proposed some fitness functions that rely on the game's outcome and are essential to detect games with serious flaws, i.e. unplayable games.
- **Completion** Tendency of a game to end with either a win or loss after a certain number of moves. Games that fail to reach a conclusive end and continue "forever" are generally less desirable.
- **Balance** A game is *balanced* if all players in each starting position have an equal chance to win the game.
- **Advantage** If the starting player is more likely to win the game.
- **Drawishness** Tendency of a game to end in a draw where no player wins.
- **Timeouts** Same as *Completion* but also considers draws as a valid ending.
- **Duration** Average number of moves necessary to end a game. It is presumed that the players' favourite game length is half the maximal game length (maximum number of moves possible).
- **Depth (Skill Level)** An indicator of skill differentiation between players is possible, i.e. a more skilled player that plans more moves ahead than a novice player should be able to win more likely. Games that rely on pure chance normally fail at this.
- **Selfishness** Refers to the concept that a player who only concentrates on the development of his pieces on the board is able to win the game. If this is the case, the game normally lacks interaction between the players.
- **Obstructive Play** Is the opposite of *Selfishness*: if a player focusses only on hindering his opponent's play may win, the game seems less desirable.

3.2 Modelling Player Experience: Quantitative Methods

- **Resilience** Measures if a game may be won by a player that only makes random moves.
- **Puzzle Potential** Ability to create puzzles within the game. With “puzzles” Browne refers here to Thompson’s definition (100) of a puzzle: “what is the best move?”.

As we pick up some of these measures in a later chapter, the interesting question arises if all of these listed factors are applicable and/or interesting to strategy games. Complexity is definitely an important factor and probably their main property. The number of different game mechanics in strategy games are usually very high, and the player has to consider several aspects of the game at the same time. The Movement item seems applicable but very important, it seems evident that different units should have different movement properties. Goal types heavily depend on the experience intended by the game designer. Multiplayer games seldom have goals besides the annihilation of the enemy. Single player campaigns, with a carefully crafted campaign might offer different goals. The branching factor correlates with the complexity of a game and is an important factor for the underlying (agent) technology as we will see in a later chapter. A large branching factor may lead to a larger response time. But turn-based strategy games usually expect players to accept a certain wait time for the computer move, therefore a larger response time is simply an inconvenience. The move effort and game tree properties may imply resp. require a certain pace of a strategy game. These properties significantly differ between real-time and turn-based strategy games. The same goes for the clarity measurements: games which require the element of fast reacting require a larger clarity, while more complexer but time-uncritical game may require the player to invest more time in planning his next moves. Strategy games have the tendency of allowing the leading player to increase its move possibilities, while the possible actions of the losing/defending player decreases. A matching example would be the concept of “map control”. The player who has more units and key positions on the map may move his units freely and may have more access to key resources than the player who practically can only move units within his base. The directly correlates to Browne’s “control” measure.

The uncertainty measure will be discussed in a later section as well as the drama indicator.

Permanence may be connected to concept of counter strategy, i.e. for every strategy player or unit type used exists a counter-strategy or counter-unit. Theoretically, all players oscillate around a power equilibrium which means that the leading player constantly changes until a decisive threshold is passed. However, quantifying “lead”

3. RELATED THEORETICAL FRAMEWORKS

and defining the leading player might not be possible at every given game state. On the other hand, once the decisive threshold is passed and a winning player is identified, the following effects could be observed by the authors in the past: competitive games are often ended by the surrender of the trailing team to escape the frustration. Non-competitive strategy games are often played until the end to enjoy the game's effects and pace, e.g. the explosions of blowing up a base.

The board coverage is usually maximised, allowing more skilled players to utilise all the objects on the map to their advantage, e.g. resources or choke points. Strategy games normally have a low “correction” value, and often contain a positive feedback loop, i.e. it is easier for the leading player to fortify his position. The balance of a strategy game is essential, and we will cover this aspect in conjunction with the “Advantage of the starting player” section 8.1.

The measures regarding the game length seem less applicable, as players are normally willing to invest a large amount of time into strategy games. The indicators regarding player interaction and selfishness are more applicable. If we compare strategy games, especially real-time, from the '90s with today's titles, mechanics have greatly improved. In earlier games so called “wall ins” were actually possible. This describes a strategy which could not be countered, where a player fortifies his base so much, that an attacking player was unable to take down any defences.

Most of Browne's measures seem applicable to our set of games as well. This is of no surprise as “combinatorial games” or generally board games have a close relationship to strategy games, as we discussed in chapter 2. Furthermore, Browne intended to model also classic games such as Chess or Checkers.

3.2.5 Learnability

Schmidhuber (101) describes “fun” as the intrinsic reward of a learning progress an agent makes. He argues that measuring a model's improvement to represent a task or environment can approximate the degree of subjective surprise or fun an agent has.

Data or information in general can be seen as a pattern or regular if it's compressible, meaning that there exists a program that predicts the data with actually using a smaller description (102). Conversely, this means that irregular noise is unpredictable and boring. If an observer didn't know a found pattern before, it may be surprising and he might be able to learn the new pattern. The learning progress then could be measured and translated into an intrinsic reward.

If an agent has the lifespan $t = 1, 2, \dots, T$ we can formulate its overall goal as the

3.2 Modelling Player Experience: Quantitative Methods

maximization of its utility (103):

$$u(t) = E_\mu \left[\sum_{\tau=t+1}^T r(\tau) | h(\leq t) \right] \quad (3.9)$$

whereas $r(t)$ is the (external) reward at the time t , $h(t)$ the ordered triplet $[x(t), y(t), r(t)]$ the sensor input, action taken and reward at the time t (hence $h(\leq t)$) is the known history of inputs, outputs and rewards until time t , and $E_\mu(\cdot|\cdot)$ is expectation operator regarding the unknown distribute μ from a set M . In other words, M reflects what is currently known about the reactions of the environment.

To maximise this utility an agent may use a predictive model p that reacts to the interactions of the agent with its environment, this means that the model depends at every time ($1 \leq t < T$) on the observed history $h(\leq t)$ so far.

We can then introduce $C(p, h)$ as the model's quality measurement. In theory this measure should take the whole history into account:

$$\begin{aligned} C_{xry}(p, h(\leq t)) = \sum_{\tau=1}^t & \|pred(p, x(\tau)) - x(\tau)\|^2 \\ & + \|pred(p, r(\tau)) - r(\tau)\|^2 \\ & + \|pred(p, y(\tau)) - y(\tau)\|^2 \end{aligned} \quad (3.10)$$

whereas $pred$ is p 's prediction of an event (103). Beside the obvious computational costs, C_{xry} doesn't take the danger of overfitting into account, that just stores the entire history. Alternatively the principle of the *Minimum Description Length* should be applied here. This takes p into account as a compressor (a program or function that compresses $h(\leq t)$ into a shorter representation) of $h(\leq t)$ so $C_l(p, h(\leq t))$ can be called p 's compression performance: the number of bits needed to specify predictors and deviations. Following Schmidhuber (101), the ultimate limit for $C_l(p, h(\leq t))$ would be $K^*(h(\leq t))$, a variant of the Kolmogorov complexity, the shortest program that computes an output starting with $h(\leq t)$. The fact that $K^*(h(\leq t))$ can only be approximated can be translated into behaviours that some patterns simply can't be learned and the reward maximizer will therefore refrain from spending too much time on these.

Since $C_l(p, h(\leq t))$ does not take computing time into account, Schmidhuber proposed to introduce a runtime dependent component that weights compression over time consumption:

$$C_{l\tau}(p, h(\leq t)) = C_l(p, h(\leq t)) + \log \tau(p, h(\leq t)) \quad (3.11)$$

3. RELATED THEORETICAL FRAMEWORKS

basically stating that one bit gained from compression is worth as much as a runtime reduction factor of $\frac{1}{2}$.

So far this has only taken the external reward, given by the agent’s environment, into account. To model something that could be seen as creativity, intrinsic motivation or fun, the reward signal $r(t)$ (according to Schmidhuber) can be split into two components:

$$r(t) = g(r_{ext}(t), r_{int}(t)) \quad (3.12)$$

where g simply maps a pair of real values, e.g. $g(a, b) = a + b$. The interesting part is the intrinsic reward $r_{int}(t)$ which is provided whenever the quality of the model improves. In its simplest case, it could be the difference of the prediction capability (with $f(a, b) = a - b$):

$$r_{int}(t+1) = f[C(p(t), h(\leq t+1)), C(p(t+1), h(\leq t+1))] \quad (3.13)$$

It should be further stated, that both models have to be trained on the same data ($h(\leq t)$) to make them comparable.

The above described theoretical framework of artificial curiosity hasn’t found much application to computational intelligence in games yet. To the author’s knowledge only Togelius and Schmidhuber himself conducted an experiment in automatic game design (104) where they have taken artificial curiosity account. A potential application would be, following the idea of *meta-learning* (105), to let several different agents play the same game, with the same rules, start setup, and probabilities regarding the reaction of the environment. Then measure their ability to “learn” the game. The learning rates could be used as a combined quality measurement of the game itself. A more unorthodox approach would be to let agents learn the game through the ruleset itself, i.e. “reading the manual” (106).

Overall, Schmidhuber’s theory, although not directly connected, seems to be a similar definition of Koster’s *Theory of Fun* (76). There, Koster says that a game is fun as long as the player can learn something new from it. Schmidhuber is less strict, and can be briefly summarised as “a game is fun if it can be learned”.

3.3 Defining Game Mechanics

Besides the definition of what is an “interesting” or “enjoyable” game, the question “what are game mechanics?” is central to our approach to procedurally generating game mechanics. This section therefore presents different approaches and arguments have been proposed in the past to coin the term “game mechanics”.

A good starting point to the definition of *game mechanics* and *game rules* might be the article “Defining Game Mechanics” by Miguel Sicart (107) from 2008. He first gives a survey over previous approaches and then discusses the definition of game mechanics as “procedures of actions” as suggested by Avedon (108) and used by Järvinen (109) using the concept of “Object Oriented Programming”.

The definition of *game rules* (in differentiation to *game mechanics*) is a contested area within game studies, but both Sicart and Järvinen seem to agree, that game rules refer to the rules applying to the gameworld itself, e.g. gravity in a physical simulation (Sicart), and that such mechanics are applied through the “gamesystem itself [...] [as] a ruleset procedure in a form of an algorithm” (Järvinen, p. 254). We interpret this standpoint as that “rules” are the rules encoded into the program while “mechanics” refers to the gameplay which emerges from these rules. Following Sicart, Järvinen suggests a classification of game mechanics into *primary* and *secondary* game mechanics (not to be confused with our definition at the end of chapter 2). This deviates slightly from the definition used by Salen & Zimmermann (110) whereas *core* mechanics are “activities that players perform again and again”. Sicart instead suggests that mechanics should be classified as “directly applied to solving challenges” (primary) and those that “cannot be used exclusively to solve the main challenges of the game, but [...] can prove of help to reach the end state of the game” (secondary).

Sicart seem to disagree with Järvinen though that game mechanics only seem to exist “so goals can be achieved”, as otherwise games without goals, e.g. *Sim City*, would not have mechanics following that definition. Sicart further states that his classification is not exhaustive: mechanics that are not related to overcoming the game’s challenge are not covered by his definition. Taking Koster’s(76) view into account, that a game should not contain inferior choices, one might argue that in an ideal game there are no game mechanics that fall out of Sicart’s definition, but psychological research has shown, that decision makers may be induced to choose a personally inferior choice (111). Hence inferior choices might be incorporated into the game design by purpose.

3.3.1 Different Layers of Rules

The *Game Ontology Project* (112) by Mateas et al. differentiates between four top-level categories, *Interface*, *Rules*, *Entity Manipulation* and *Goals*, and a large number of sub-entries; what Aarseth criticises as “less than intuitive” (113). Aarseth instead suggests dividing the gameplay (as a process) into two different layers: “The semiotic layer [...] is the part of the game that informs the player about the game world and the game state through visual, auditory, textual and sometimes haptic feedback. The

3. RELATED THEORETICAL FRAMEWORKS

mechanical layer [...] is the engine that drives the game action, allows the players to make their moves, and changes the game state.” According to Aarseth, the mechanical layer also incorporates the mechanics of objects interacting with the world.

Salen and Zimmermann (110) decompose game rules into three layers:

- Implicit Rules
- Constitutive Rules
- Operational Rules

Implicit rules Implicit rules in this context mean “unwritten rules” (e.g. players may not cheat, players shall not overthrow the board or younger players may take back a foolish move). While those are definitely important to the social aspect of play and what makes them enjoyable, they are less interesting from a computer science perspective as they can’t be formalised (otherwise they would become explicit rules) and are therefore not fungible for any algorithm.

Constitutive rules Constitutive rules are the underlying meta-model a game is about. This layer lines up with Sicarts definition of “game rules”. Evolving game rules instead of mechanics could be called a top-down-approach. Salen and Zimmerman give the example of *Tic-Tac-Toe* and *3-to-15* which share the same constitutive rules. We assume that the reader is familiar with the game of Tic-Tac-Toe, and the rules of *3-To-15* can be taken from *Rules of Play* (110, 128):

1. Two players alternate turns.
2. On your turn, pick a number from 1 to 9.
3. You may not pick a number that has already been picked before by either player.
If you have a set of exactly 3 numbers that sum up to 15, you win.

The explanation of the analogy can be found on the same page: *3-To-15* is a “magic square puzzle”, where any horizontal, vertical, or diagonal sums up to 15. By playing a game by the rules of *3-To-15* players actually play a game of *Tic-Tac-Toe*, say Salen and Zimmerman.

Operational rules Operational rules conform with what Sicart defines as “game mechanics”, the actual “written-out” rules that come with the game. They define the legal moves and how entities within the game may interact.

3.3.2 Games as Systems

Salen and Zimmermann also see games as a system (110, 49f) and they describe the parts of the games as “objects” (e.g. chess pieces), “attributes” (e.g. the color of a piece) and “internal relations” (e.g. a white piece may remove a black piece). This is actually the common paradigm for object oriented programming and the same idea as Sicart’s (107). Another point that is raised by Salen and Zimmermann (110, 44) is the idea of “interpretation of signs”, what is basically the core idea of the field symbolic AI: the differentiation between a sign (symbol, syntax) and its meaning (semantic). The argument here is that this is not necessarily pre-defined and is hugely influenced by the surrounding system (may be the game or even a cultural context). Although this a very narrow summary of this important topic, it is an important argument to be able to detach the game’s system (the constitutive rules) from the actual implementation (sometimes called “theming” or “audiovisual presentation”).

Järvinen (109) extends the idea of games being systems and proposes a taxonomy and library of game mechanics. He defines game mechanics as “verbs” that could be used to describe the (re-)actions of players within the games. In a more technical sense, this could be seen as transitions between game states. The problem with this top-down approach is, like Järvinen states himself, that is not exhaustive and may fail to classify/categorise unknown game mechanics or nuances of exiting ones. Using verbs as a representation therefore doesn’t seem to be applicable for evolving game mechanics through algorithms.

3.4 Summary

This chapter presented a large variety of methods which have been previously proposed or used to model and measure the experience a person might have while playing a game. Several frameworks have been proposed to capture and analyse the experience. We have presented a number of qualitative methods which mainly focus on the emotional state of the player, and a number of qualitative methods which seem to focus rather on properties of the game itself rather than the person playing it. However, the collected data does not stand out alone, but is in return used to infer a player’s emotional state. We also presented related literature regarding the term “game mechanics” and how game rules may be formalised. While this chapter mainly covered related theoretical research, the next chapter will focus on related applied research from the field of computational intelligence.

3. RELATED THEORETICAL FRAMEWORKS

Chapter 4

Related computational intelligence in games research

While the the previous chapter focussed on related work on frameworks regarding capturing the experience of “gaming”, this chapter presents more technical work which is connected to the process of creating content, machine learning, and previous approaches to the modelling of game rules.

4.1 Generating Content

We start with techniques and concepts of automatic creation of artefacts through algorithms. The term “artefact” here refers to a resource which is not the program’s code itself. In the context of computer games the term “content” is used synonymously. This section presents work that can be categorised as procedural content generation, but does not cover the generation of game mechanics. Game rule generation is explicitly discussed in section 4.4.

4.1.1 Search-based procedural content creation

Search algorithms are (as the name states) algorithms which search a solution for a given problem. The problem is often implicitly defined by a function which determines how good a solution is in terms of the problem. The task of the algorithm is then to find the optimum, or at least a solution that is sufficiently good. The space of potential solutions is often multi-dimensional and commonly referred to as the *search space* of potential solutions. There exists a wide variety of different search algorithms and strategies, a few well known examples are *Local Search* (114) or *Tabu search* (115).

4. RELATED COMPUTATIONAL INTELLIGENCE IN GAMES RESEARCH

Different search-based algorithms employ different strategies to traverse the search space in order to find the optimal solution most efficiently.

Coming from the angle of procedural content generation (PCG), Togelius et al. introduced a taxonomy of search-based content generation (SBPCG) in games in 2010 (3). They defined *game content* as all aspects of the game that affect gameplay but are not NPC-behaviour or the game engine itself. Among the examples they give are maps, levels, weapons, and quests. They propose a taxonomy with five different dimensions:

- **Online versus Offline** a PCG system can produce new content online (at the time the game is actually running) or offline (before the game starts).
- **Necessary versus Optional** this category doesn't refer to the PCG system itself but to the content it creates. Togelius et al. define necessary as everything in a game that is needed to complete the game. Optional content therefore may be non-functional (although this is an undesirable) while necessary content needs to be correct.
- **Random Seeds versus Parameter Vectors** this addresses the problem of representing PCG content in its compact state (genotype).
- **Stochastic versus Deterministic Generation** classifies if the generator will produce the same artefact every time it's invoked with the same input vector.
- **Constructive versus Generate-and-test** Togelius et al. compare generators that create assets and "be done with it" with generators that evaluate and potentially reject a solution.

Especially the last point, the evaluation of the generated content, is essential for a search strategy. A content generator may iteratively improve a solution (or a whole set of candidates) until the required quality is reached.

We can also extract the core motivations of employing PCG in games from the same publication. Togelius et al. enumerate:

- **Memory consumption** this refers to the game data shipped with a game. Procedurally generated content generally takes less space until it is "expanded".
- **Expenses of manually creating game content** algorithmically generating game content or variations of it decreases the amount of labour required to create it.

- **Emergence of new types of games** if content is created online, and the creation process is driven by the playing style of a particular player, the game may change in an unforeseen way
- **Augmentation of the imagination of the game designer** similar to the emergence of new game types, the PCG system can inspire its creator/user.

Although our work is primarily intended for offline use, it could be theoretically possible to change a game’s rules while the player is playing it. However, ignoring the game design implications, this seems impractical due to the long time it takes to run the evolutionary process; as the framework is clearly on the “generate and test” side. The SGDL framework, introduced in chapter 5, is “deterministic” since SGDL trees are always interpreted by the engine in the same way. As the genotype models game mechanics on a rather verbose level, we argue that SGDL corresponds to the “parameter vector” side in Togelius’ work. The last dimension, if the generated artefacts are necessary or optional, depends on if SGDL is use to model all game mechanics or just parts of it.

4.1.2 Procedural Level Generation

Probably the most explored aspect of procedural content generation is the generation of “levels”. The term “level” here can be extended to track, stage, map, etc. or more generalised as “world”. The generation of game worlds (as in: topological structure) is probably the most common application of procedural content generation in commercial games, and a fairly well-studied problem. Commercial examples range from galaxies in the first *Elite*, to newer games with aspects procedurally generated levels, such as *Diablo III*, or games which are heavily based on the procedural generated worlds such as *Minecraft* or *Dwarf Fortress*.

As stated above, there have been many projects on procedural level generation. Examples range from levels for *Super Mario Brothers* (116) or similar (117), to racing games tracks (118), first person shooter levels (119), and strategy game maps (120). A rather abstract approach was presented by Ashlock in 2010: the generation of mazes using a color-coded grid (121). The player could only traverse cell boundaries that had colors also adjacent in the chromatic circle.

4.1.3 Interactive Storytelling

A separate - but related to rule generation - field of procedural content generation is the construction of coherent stories and plots. Even though these approaches do not

4. RELATED COMPUTATIONAL INTELLIGENCE IN GAMES RESEARCH

address the generation of game mechanics, they seem interesting as they also address the aspect of content generation. Unfortunately no work known to the authors has been done on stories for strategy games, but we believe that the presented approaches could be adapted to tactical gameplay.

One prominent example is the *Faade* system, an experimental game published by Mateas et al. (122) in the early 2000s. Players may interact with the system through the input of natural language, and affect a story framed as a dinner event with a married couple. Depending on the input (or lack thereof) of the player, the story may take different turns and arrive at different endings. The *Faade* system is generative in the sense that it mixes and sequences behaviours in sophisticated ways, but it does not generate the individual behaviours (ibid. p. 4). Instead, a component called the *drama manager* picks so called *beats* from a collection of behaviours tailored to a particular situation. The choice is based on the current state of all objects and characters in the game. The drama manager selects new beats from the set of potential beats based on a “tension” value, ultimately forming an overall story arc.

Another approach was published by Cheong and Young (123). Their computational model of narrative generation of suspense uses planning algorithms to change the natural order of events of a story (fabula; all the events in the order they take place) into a partial ordered sequence, the *sjuzhet*. The *sjuzhet* is the narrative order of the story, may differ from their natural order, or may be just a subset, i.e. a story may be told in flashbacks or flashforwards, or certain plot elements may be hidden from the reader resp. viewer. Cheong and Young used a heuristic for *sjuzhets* based on suspense. The suspense level of the reader was measured based on a notion articulated by Gerrig and Bernardo (124) which sees an audience as problem solvers: an audience feels an increased measure of suspense as the number of options for the protagonist’s successful outcome(s) decreases. Therefore the suspense level of the reader can be defined as the the inverse of the number of planned solutions for the protagonists’ goal.

4.1.4 Cellular automata

Cellular automata are not a technique especially developed for procedural content generation. They are used to model spatial discrete dynamical systems, where the state of each cell at the time $t + 1$ depends on the state of its neighbouring cells at time t . Cells can be ordered in different spatial structures (e.g. one dimension line, two dimensional grid, etc.). Cellular automata were initially developed by Ulam and von Neumann (125) to model robots that could assemble new copies of themselves. Like conventional finite-state machines they consist of a set of states and a transition function. Since each cell

in the grid can take different states (such as *On* and *Off*) there exist c^s states in the automaton, where c is the number of cells and s is the number of different states a cell can take. Furthermore the transition function requires a *neighbourhood*, i.e. which cells' states affect the determination of a cell's next state. The 4-cell von Neumann neighbourhood (126) and 8-cell Moore neighbourhood (127) are probably the most used ones. The first published automaton had 29 states and could reproduce a given pattern indefinitely.

In the 1970s a different application became well-known: Conway's *Game of Life*, in which each cell in a two-dimensional grid was considered either as *dead* or *alive* and neighbouring cells became alive if three "living" cells were adjacent. There are multiple interpretations of the *Game of Life*: biological evolution, physical mechanical systems or energy and matter in chemistry. A more playful application developed from the challenge of creating self-reproducing patterns like *Gosper's gliding gun* (128) that shoots little spaceships in a reoccurring pattern.

While the applications mentioned can be more seen as games *inside* cellular automata, research has also been conducted into how to use them as part of games or to create game assets. A recent application was the creation of levels for roguelike games by Johnson et al. in 2010 (129). Each cell represented a part of the map while its multi-dimensional state defined different features within the game, e.g. walls, rocks or different special areas in the game. Sorensen and Pasquier (130) used a similar block representation for the generation of levels for *Super Mario Brothers* but used a genetic algorithm instead an automaton for the generation itself.

4.1.5 L-Systems

L-Systems (named after the Hungarian biologist Aristid Lindenmayer) were originally introduced in 1968 (131) as a model to describe the growth and branching of plants. The rules were developed to describe things such as symmetry of leaves or the rotational symmetry of flowers (132).

The core concept of L-systems is *rewriting*, where a part of the initial object is replaced with a new object or extended rule or *production*, similar to the concept of *functions* or *recursion* in any modern iterative programming language. While Chomsky's work on formal grammars (133) was similar, Lindenmayer-systems are intended primarily as a solution to produce geometrical shapes.

DOL-Systems The simplest L-system is a so called DOL-System, a deterministic context free L-system for string representations. It is defined as an ordered triplet

4. RELATED COMPUTATIONAL INTELLIGENCE IN GAMES RESEARCH

$G = \langle V, \omega, P \rangle$ whereas V is the set of literals (alphabet), $\omega \in V^+$ a set of axioms that are the distinguished start strings, and P a finite set of production rules as $P \subset P \times P^*$. Rules $(a, \chi) \in P, a \in V, \chi \text{ in } V^*$ are denoted as $a \rightarrow \chi$. The application of rules, the process of creating, is called *derivation*. It is assumed that for each $a \in V$ exists one production rule. If no production rule is explicitly stated, the rule $a \rightarrow a, a \in V$ is assumed. It is further assumed that for every $a \in V$ only one $\chi \in V^*$ exists, otherwise the system would not be deterministic.

Example Consider the DOL-System $G = \langle a, b, \omega, a \rightarrow b, b \rightarrow ba \rangle$. If we let $b \in \omega$, then the following derivation would unfold: For the sake of completeness it should be added that there also exists more advanced variants of L-Systems (LS) such as Stochastic-LS, where rules are only applied with a certain probability, Context-sensitive-LS, where the left side of production rules may also be $\chi \text{ in } V^+$, and Parametric-LS that extend the triplet with another set Σ of formal parameters that are used to evaluate logical expressions during the derivation. A similar concept are Functional-LS (134), where the terminal symbols are replaced by terminal functions, allowing postponing resolving requirements. The latter were used by Martin et al. for procedural scenario creation in serious games (135), what may serve as a good example for a L-system application in games that is not involved in the generation of art.

4.2 Computational Creativity

Work in this interdisciplinary field concerns problems such as how to quantify “creativity”, if it is possible to create an algorithm or machine which is capable of being creative on a human level, and whether a program or machine could replace or augment a person’s creativity. Some key concepts around “what is creativity?” have been proposed in the past. Following the idea that being creative means “to create something new”, Margaret Boden proposed (136) to distinguish between psychological- and historical-creativity (abbreviated as P- and H-Creativity). P-Creativity refers to creating something that is new to the creator, while H-Creativity refers to creating something that has been never created before by any human. Boden further proposed the terms “exploratory” and “transformational” creativity. Although an object may be *creative* by being “new”, nothing is created without a context. If an object is different, but does not differ in the *number* and *type* of features (here: conceptual space, but could be also compared to the *search space* described in section 4.1.1) to what it relates to, it is created through “exploratory creativity”. Objects which were created through changing the conceptual space are “transformational creative”, says Boden. We argue,

that these two concepts can be found in games published as well if we understand the term “genre” as conceptual space. Although exploratory games, i.e. games that take existing game ideas or themes (“clones”), are often received more negatively. Transformational creative games are often game ideas which mediated from other media, especially in the 1980s e.g. pong from table tennis, or hybrids between two existing genres.

Coming back to field of computational creativity, Ritchie stated in 2007 (137), that “creativity of humans is normally judged by what they produce” and that “underlying processes are not observable factors, hence not reliable [for assessing a person’s level of creativity]”. He also points out, that there is danger of a circular argument if we assess both, which might lead to “the artefact is creative because it is the result of a creative process because the result is creative”. Instead the process of creation (the method) should be considered an artefact itself. Colton picks up on the idea, that process and created artefact should be treated equally, and motivates this with an example of an art lover who appreciates one of two identical pictures more because of details of their creation processes. One picture is more appreciated than the other because of a different background of the artist who created it. Similar behaviour can be found among computer games players, who are sometimes more beneficial towards a game made by an indie developer than a game made by one of the major publisher, a game with technical shortcomings, less content, or non-standard graphics is received differently depending on the game studio who created it. Colton proposes a “tripod model” (138) to assess the behaviour of software, whether it should be perceived as “creative” or not. Although Colton focusses more on software of the visual arts (e.g. his own “The Painting Fool” project (139), an artificial painter), we believe that this might be applicable to procedural content generation for games as well. Here, the game designer uses or creates an algorithm to create assets (e.g. textures, 3D models, etc.). These are used in a game which ultimately will be consumed by players (the audience). In Colton’s tripod model, each leg represents one of the behaviours which are required in the overall process: appreciation, imagination, and skill. As there are three parties involved, (the programmer, the software, and the audience), each can contribute to the process. Therefore each leg of the tripod has three segments which can be extended based on the size of the contribution. Then, says Colton, a software system can be perceived as “creative” if at least one segment of each tripod leg is extended, i.e. if the algorithm contributes skill, imagination, and appreciation, it can be perceived as “creative” regardless the actions of the programmer and the audience. After a series of case studies presented in the paper, Colton points out that “software which we want to

4. RELATED COMPUTATIONAL INTELLIGENCE IN GAMES RESEARCH

ultimately be accepted as creative in its own right needs to subvert any given notions of good and bad artefacts.“ His tripod model is intended to “level the playing field somewhat when people assess the value of computer generated artefacts.”, i.e. make the creativity of humans and artificial creators comparable by “providing consumers of the artefacts with some high-level details of how the software operates.”

4.3 AI and Learning in Games

The field of procedural content generation borrows various techniques from the field and artificial intelligence or machine learning. Furthermore, our work on general game-playing makes use of algorithms and concepts from this field. This section presents techniques which were used throughout our experiments along with other related work. We utilised different techniques to implement agents which are able to play a range of games expressed in SGDL. Ultimately, our agents should be able to sufficiently play any game expressed in SGDL, given that they had enough time to learn its rules. Techniques presented in this section are not exclusive to games research, but are discussed in this context.

4.3.1 Game Tree Search

There has been extensive research done on AI for traditional board games. In particular, Chess has figured prominently in AI research from the very start, as it is easy to formalise and model, and has been thought to require some core human intellectual capacity in order to play well. Among prominent early attempts to construct chess-playing AI are Turing’s *paper machine* (140) and McCarthy’s *IBM 7090* (141). Both of these used the MinMax algorithm, which builds a search tree (or: “game tree” hereafter) of alternating actions of both players up to a certain depth (*ply*) and estimates the value of the resulting board configurations at the nodes of the tree using an evaluation function. This poses the question of how to construct an accurate evaluation function. An early pioneer in using machine learning to construct evaluation functions was Samuel, whose self-learning Checkers player anticipated the concept of temporal difference learning (142).

Advances in both algorithms and computer hardware permitted a program built on the MinMax idea to win over the human Chess world champion in 1997 (143). Subsequently, much research on board game AI shifted to the considerably harder problem of playing the Asian board game Go. Go has a much higher branching factor than Chess, and it is also harder to construct a good board evaluation function, meaning

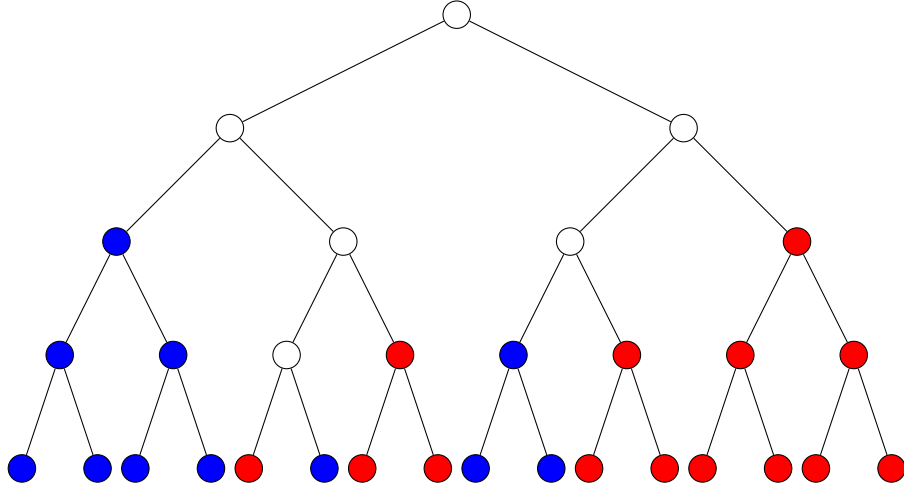


Figure 4.1: Game Tree of an arbitrary two-player game. Each turn (node) the current player has to make a binary decision (vertex). Blue nodes indicate a path through which one player will certainly win; red nodes analogously for a second player. White nodes indicate game states where the game is not decided yet.

that MinMax-based approaches have so far performed very poorly on Go. The current best AI approaches to Go are instead based on Monte Carlo Tree Search (MCTS), a stochastic technique that does not normally use an evaluation function (15, 144). Both MinMax and MCTS will be described in the following.

The basic idea of a “game tree” is to represent possible courses of moves (for all players) in a tree form, i.e. for each state in the game there exists a node, each vertex to a child node represents a possible decision of the active player at that state. More possible actions per turn result in a higher branching factor. Graphical examples of game trees are commonly given for two player games, but the method is not restricted to such. The number of players in a game does however effect the search strategy for the best move of the current player, and the presented algorithms in this section work on trees for two player games. The purpose of game tree algorithms is normally to find a path through the game tree that will lead the current player most likely to a winning outcome. Algorithms often rely on stochastic methods, as games may contain aspects of chance. Another aspect to consider is the branching factor of a game tree, directly increasing the runtime complexity when evaluating an exponential number of possible outcomes. Figure 4.1 illustrates the course of an arbitrary two-player game, with each node coloured in a players colour that, if the player follows that path, would lead to a guaranteed win. This method is called “retrograde analysis”.

4. RELATED COMPUTATIONAL INTELLIGENCE IN GAMES RESEARCH

4.3.1.1 Min-Max Search

To create a full game tree (where every leaf node actually represent an end state of the game) is very impractical for games which are not trivial. A higher branching factor combined with increased game length leads to a tree which is infeasible to traverse and compute in reasonable time. An approach to overcome this problem can be found with the “Min-Max” algorithm. The algorithm “Min-Max” (sometimes Minimax) was originally designed for two-player zero sum games (145), where players take alternative turns. The name is derived from its strategy of “maximising its own player’s outcome, while minimising the other’s”.

Instead of generating a complete game tree, the Min-Max algorithm creates a tree of a certain search-depth (synonymously: look-ahead). Each leaf node is assigned a quantitative value corresponding to the standing of the player who is invoking the algorithm (player A). The values are back propagated through the intermediate nodes up to the root of the tree. The value of an intermediate node is chosen as follows: if the node represents a turn by player A, the largest value from the child nodes is assigned. If it is a move by player B, the smallest value from the set of child nodes is assigned. The child of the root with the highest value is therefore the most promising move for the current state and therefore will be picked by the algorithm. Figure 4.2 clarifies this on the basis of an example.

The advantage of MiniMax is clearly its versatility, e.g. Chess and Go can both be played by the same algorithm. If the decision tree is created with an infinite look-ahead, i.e. each leaf node is actually a state where the game ends, the algorithm requires no knowledge about the game to make a qualified decision apart from the information that a player has lost or won. However, the amount of memory resources and required computation time make a limitation of the look-ahead necessary for almost all interesting games. Then, the skill of the artificial player depends on the heuristic function to assign a value to each leaf node. The main challenge here is to design a meaningful heuristic function. If it relies on features which are actually irrelevant to winning the game, the artificial player will display a poor performance in terms of winning the game. Another disadvantage in terms of runtime performance is that the MiniMax algorithm explores every child of a node regardless of how (un)promising the already evaluated siblings of the child node are. This results in many evaluations of nodes which the algorithm most likely would never chose.

One approach proposed to limit the search to promising branches of the tree is the “Alpha-Beta” extension for MiniMax (also called “Alpha-Beta-Pruning”). The idea is that the algorithm tracks two values (the eponymous “alpha” and “beta”) during the

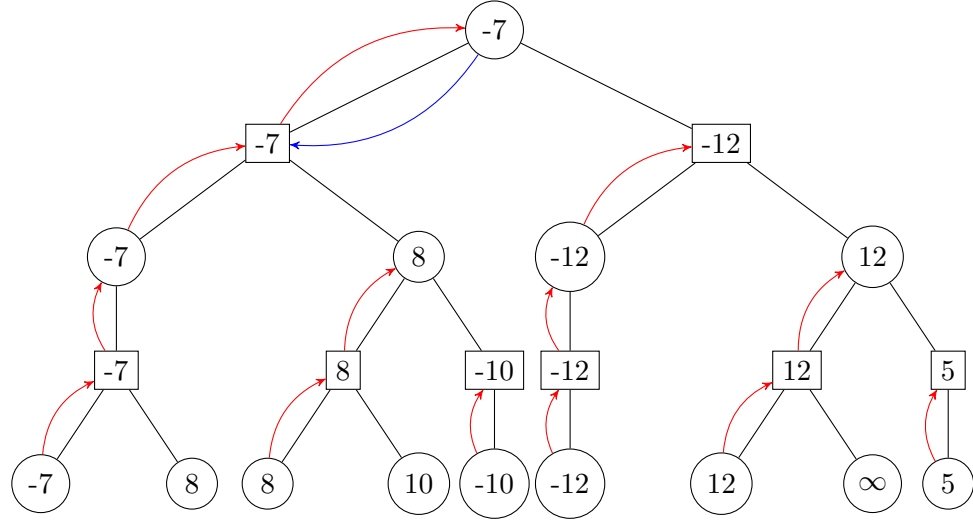


Figure 4.2: Example for the Mini-Max algorithm. The algorithm creates a game tree, and assigns each leaf node a value based on the standing of player A. The value “infinity” (∞) is often chosen to represent a game won by player A (resp. $-\infty$ for losing the game). The red arrows indicate how the values are back propagated through the tree: rectangular node select the minimum value from their children, circular nodes the maximum. The blue arrow indicates which move the algorithm ultimately selects.

search: alpha is the value player A would at least reach, beta the value player B would reach at maximum (note that in this algorithm the value of player B is to be minimised). If a maximising node (player A) holds a value which is higher than beta (beta-cutoff), the exploration of that node will be stopped because player B would have never played the move leading to this node. If instead the value of the node is larger than the current alpha value, alpha is raised to that value and the exploration continues. The algorithm proceeds analogously for minimising nodes (player B): branches with values smaller than alpha are not explored (alpha-cutoff), values smaller than beta are explored and beta lowered respectively.

4.3.1.2 Monte-Carlo Tree Search

As discussed about the Min-Max tree search, some games have game trees with enormous ply depth or branching factors. This is especially true for strategy games, as there are normally a large number of units present on the map, each with a multiple number of possible actions. The number of possible outcomes for the next n moves can be approximated as n^{x^y} , with x the number of units on the battlefield, and y the number of possible moves per unit. This assumes that all units have one action to perform,

4. RELATED COMPUTATIONAL INTELLIGENCE IN GAMES RESEARCH

and that all units have the same number of possible actions, and that both players have the same number of units on the map. Despite the assumptions, the increased computation time renders a traditional Min-Max approach infeasible. Furthermore, a heuristic function for a strategy game might either be hard to design or expensive to compute.

In these cases the application of a Monte-Carlo techniques may be beneficial. In a Monte-Carlo tree search (MCTS) the heuristic function is replaced by a sampling technique: in each designated leaf node the game state is cloned n times, and each copy is played out until the end by selecting random actions for each player. This eliminates one of the main drawbacks of Min-Max: the design of a meaningful heuristic function when an exhaustive search is not possible. MCTS has also the advantage that selecting random actions significantly faster and requires less resources than informed decision making. The utility of each leaf node for each player is determined by the percentage of wins over the number of randomly simulated games (referred to as “roll-outs”). Back propagating the utility of each leaf node, weighting the intermediate nodes in the process, leads to the selection of the currently most promising course of actions for a player. Another difference to Min-Max is, that the search tree is not built equally. Different strategies exist to explore only the most promising tree branches while also maintaining a certain diversity.

Monte-Carlo Tree Search in games has recently gained some popularity with the board game Go (146). In regards of strategy games, the work of Silver et al. with *Civilization II* may be highlighted (106, 147). They replaced the rollouts with an artificial neural network. An interesting note about the presented agent is also that the state evaluator function was trained through linguistic knowledge of the game’s manual. A general overview of the applications of MCTS in games has been published by Browne in 2012 (148).

MCTS is a technique used by one of our general gameplaying agents (presented in section 7.2.5. One of the aspects explored there, that makes MCTs significantly different from MinMax, are different exploration strategies, i.e. strategies of how to expand the search tree. The Monte-Carlo value (Q) can be seen in equation (4.1), where Γ is an indicator function returning 1 if the action a was selected in position s at any of the i steps, otherwise 0, $N(s, a)$ is the number of simulations through s where action a was chosen, and $N(s) = \sum_{i=1}^{|\mathcal{A}(s)|} N(s_i, a_i)$, where $\mathcal{A}(s)$ is a finite set of legal actions from state s .

$$Q(s, a) = \frac{1}{N(s, a)} \sum_{i=1}^{N(s)} \Gamma_i(s, a) z_i \quad (4.1)$$

Several tree policies have been published such as UCT (149) in Eq. (4.2), progressive bias (150) in Eq. (4.3), Monte-Carlo Rapid Action-Value Estimation (MC-RAVE) in Eq. (4.4), (4.5) and (4.6) and UCT-RAVE in Eq. (4.7) (151, 152).

UCT solves the exploration dilemma by utilizing the UCB1 (153) algorithm by scaling the exploration factor c , so the amount of exploration can be limited.

$$Q_{UCT}(s, a) = Q(s, a) + c \sqrt{\frac{\log(N(s))}{N(s, a)}} \quad (4.2)$$

Progressive bias is an added heuristic to the standard UCT heuristic to guide the search. The impact of the heuristic lessens as simulations through state s using action a increase.

$$Q_{pbias}(s, a) = Q(s, a) + c \sqrt{\frac{\log N(s)}{N(s, a)}} + \frac{H_{(s,a)}}{N(s, a) + 1} \quad (4.3)$$

The RAVE values in MC-RAVE quickly converges to a bias value $\tilde{Q}(s, a)$ for action a from the subtree of the node representing state s . Since this value is biased, MC-RAVE uses a decreasing factor $\beta(s, a)$ relying on a k -value to determine how fast the factor decreases. Sylvain Gelly and David Silver found the highest win rate in Go using a k -value of 3000 (151). Due to lower MCTS iterations, the k -value had to be lowered in the experimentations and rely more on the actual Monte-Carlo values and not the biased RAVE-values. Because the MCTS agent used a heuristic, the biased RAVE-values were evaluations from subtrees instead of actual playout values.

$$Q_{MCRAVE}(s, a) = \left(\beta(s, a) \tilde{Q}(s, a) + (1 - \beta(s, a)) Q(s, a) \right) \quad (4.4)$$

$$\beta(s, a) = \sqrt{\frac{k}{3N(s) + k}} \quad (4.5)$$

$$\tilde{Q}(s, a) = \frac{1}{N(s, a)} \sum_{i=1}^{\tilde{N}(s)} \Gamma_i(s, a) z_i \quad (4.6)$$

UCT-RAVE adds the exploration factor $c \sqrt{\frac{\log N(s)}{N(s, a)}}$ from UCT to MC-RAVE.

$$Q_{UCTRAVE}(s, a) = Q_{MCRAVE}(s, a) + c \sqrt{\frac{\log N(s)}{N(s, a)}} \quad (4.7)$$

4. RELATED COMPUTATIONAL INTELLIGENCE IN GAMES RESEARCH

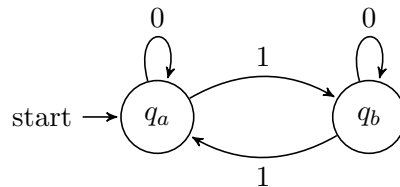


Figure 4.3: Example of a simple finite state machine. The input 1 lets the machine alternate between stating state q_a and state q_b . The input 0 has no effect on the current state.

4.3.2 State machines

Finite state machines (or finite automata) are a widely used mathematical model used to specify behaviour of a (software) system. The basic mathematical model of a finite state machine (FSM) is a quintuple, including an alphabet Σ of possible inputs, a non-empty set of states S that can be visited during program execution, the initial state s_0 , a state transition function $\delta : S \times \Sigma \rightarrow S$, and a set $F \subset S$ of final states which may be empty. A FSM can be extended to a finite state transcoder (FST) by adding a non-empty set of output symbols Γ and changing the transition function to $\delta \subseteq S \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \times S$ (where ϵ is an empty symbol). Figure 4.3 shows a simple example, a state machine which alternates between two states without any output. The set of states S is here $S = \{q_a, q_b\}$ with an input alphabet $\Sigma = \{0, 1\}$. The list of final states in this example is the empty set $F = \emptyset$. The state transition function δ can be described as a set of triples, describing the start state, input, and a new state $\delta = \{(q_a, 0, q_a), (q_a, 1, q_b), (q_b, 0, q_b), (q_b, 1, q_a)\}$.

It should be added, that one of the disadvantage of traditional FSM is that with increased complexity of a program, and therefore increased number of states necessary to represent it, the number of state transitions grows exponentially if fully connected. This ultimately leads to graphical representations which may be very difficult to comprehend for humans. One approach to this problem is the concept of “hierarchically nested states” (or “hierarchical state machines”) which is part of the Unified Modelling Language (UML). Every nested state is also implicitly its surrounding state. If an input is not explicitly handled by the current state, it is passed to a higher-level context, the superstate.

Evolving FSMs for various problems goes back to the 1960s to work by Lawrence J. Fogel et al. (154) and has been established as a widely applied technique in the field of evolutionary computation. The evolution of FST was recently explored by Lucas in 2003 (155).

4.3.3 Decision- and Behaviour Trees

Decision Trees are graphs which can be used to visualise algorithms. They represent a chain of *if ... then .. else* decisions. The root node represents an initial problem, e.g. should a game character move left or right, and each node represents an atomic decision like “if $x > 5$ then follow the right branch, follow the left branch otherwise”. Leaf nodes represent answers to the initial problem, e.g. “move left” or “move right”. Besides behavioural decisions, they are widely used for classification (156).

Behaviour trees are a concept originally published by Domey (157) to visualise software requirements written in natural language in an more comprehensible form, but have been used for (game) AI development in the past (158, 159). Their advantage over more traditional AI approaches, e.g. finite state machines, is their simple and reusable modularity. Each tree describes potential paths from an initial- to one or more goal-states and thus can be linked with other trees to achieve high-level goals. Behaviour trees are probably the most similar technique compared to SGDL (even though for a completely different purpose): behaviour trees distinguish between two types of nodes: conditions and actions. While conditions query a program state steer the control flow, i.e. which nodes of the tree are traversed, actions execute methods and can change the program state. The latter distinguishes them from decision trees.

So far, behaviour trees have been used mainly for modelling game bot behaviour. Lim et al. (160) have used behaviour trees to evolve the behaviour of a game bot for *Defcon*, a commercial real-time strategy game. Their bot was able to outperform the built-in AI in more than 50% of the games played. Two of the authors of that publication, Baumgarten and Colton, had already published a bot for *Defcon* earlier (16). There, they used a combination of case-based reasoning, decision tree algorithms and hierarchical planning. Unfortunately, no comparison with the behaviour tree bot are known to the authors.

4.3.4 Neural Networks

(Artificial) Neural Networks are a commonly used concept in machine learning. They are a mathematical model, inspired by biological neural network, used for function approximation and pattern recognition. Initial research was performed by McCulloch and Pitts (161). Other significant publications were made by (among others) Kohonen (162), Werbos (163), and Hopfield (164).

The overall idea is based on the concept of a *perceptron*, a single node in a feed-forward fully connected graph. Each perceptron has an activation function which gen-

4. RELATED COMPUTATIONAL INTELLIGENCE IN GAMES RESEARCH

erates one single output based on n different inputs. The inputs are combined into a weighted sum and fed into an activation function which determines the output of a perceptron. Perceptrons are arranged in layers. The first layer takes several inputs from an external source, the last layer forms one or multiple outputs of the network. The layers in between are normally referred to as “hidden layers”.

Neural Networks are able to approximate and “learn” unknown functions. In a supervised learning process, inputs are fed into the network and the error from the (known) output values are back propagated (hence the name “back propagation” of the learning process) into the network. This alters the threshold parameters of the activation function or the weights of inputs in each perceptron. Back propagation however does not adapt the topology of a neural network. This may be beneficial if we consider that many computer strategy games have enormous branching factors. Since the high branching factor on the micro decision level is very challenging for AI systems, it may be more promising to limit their usage to macro level decisions and rely on simpler techniques, e.g. fixed finite-state machines, for actual task executions.

Olesen et al. used Neuroevolution of Augmenting Topologies (NEAT). NEAT, credited to Stanley and Miikkulainen (165), and neuroevolution in general treat neural network as a genome of a genetic algorithm, and uses the principle of evolution to evolve a network topology which approximates the required function. In regards to strategy games, NEAT has previously been used to evolve agents for the game *Globulation 2*. Through dimensionality reduction based on expert domain knowledge they were able to create controllers that could adapt to a players’ challenge level offline and in realtime (166).

4.3.5 Genetic Algorithms

A Genetic algorithm (GA) is search heuristic that mimics natural evolution. GAs are classified and often used synonymously as Evolutionary Algorithms (EA). Although related, the field of EA distinguishes between GAs and related techniques such as Genetic Programming (see section 4.3.6) and others. Using natural selection, and other mechanisms related to natural evolution, as a search heuristic goes back to the 1950s when a publication of Nils Aall Barricelli (167) et al. received a lot of attention. Today GAs are a popular technique for optimisation and search problems.

A Genetic Algorithm consists of a problem statement, formulated as a fitness function, that evaluates the quality of a solution. The solution is often encoded in a genotype (or chromosome) which is resolved into a phenotype later in the testing process. It may be clearer when this process is translated into an example: the template for

most living beings (e.g. humans) are their DNA. This DNA acts as a template for the features the body will grow (e.g. skin color or body height); environmental factors set aside. Beside the problem encoding and the fitness function, a GA also consist of several mechanism for how new solution candidates are created. They can be created through pure random sampling or based on the fitness of previously tested solutions. Solutions can be either created through combining two previous solutions (cross-over) or manipulating a single one (mutation). This on the other hand creates the problem of which previous solutions should be selected to form a new solution. A common solution for that is selecting candidates from the top 50% of the previous generation or do a weighted-random-selection among all candidates.

Overall a genetic algorithm improves a set of possible solutions iteratively, where each iteration is normally referred to as a “generation”. Commonly each generation is constructed based on the previous one as described in the paragraph above. The algorithm continues until a stop condition is hit. Usual stop conditions are time constraints (number of iterations), or reaching the goal (optimal or “close enough” fitness).

4.3.6 Genetic Programming

Genetic Programming (GP, we will use this synonymously for the field and a potential solution (Genetic Program) in the following) is a evolutionary technique that is related to Genetic Algorithms (GA). While early GPs were used to model finite-state-automata or markov-decision-processes, the “modern” GP is a technique that uses a tree based representation to model logic statements or programs in general. Trees are often used to model mathematic expressions, but may also contain functions which have side effects (analogue to imperative and functional programming). This approach was developed by Cramer (168) in the 1980s, but became widely popular through the work of Koza (169) later. Before tree-based GP is explained further, it may be mentioned that also the technique of *linear genetic programming* exists where chromosomes are interpreted as a sequence of commands, similar to assembler instruction codes that are sequentially fed into a microprocessor. Furthermore, *Cartesian genetic programming* is a technique developed by Miller and Thompson (170) to encode complete graphs into a chromosome.

However, the form of GP relevant for SGDL uses a tree based genotype. There is no standard encoding but a common technique is to use a string-based representation. For example, the string $y = 3 * x * x + 4 * x + 12$ is a common way to express the equation $y = 3x^2 + 4x + 12$. If the mathematical operators are interpreted as intermediate nodes and the constants as leaf nodes, the example may be also noted as a tree: While the parent-selection is no different from GAs, the cross-over mechanism refers to changing

4. RELATED COMPUTATIONAL INTELLIGENCE IN GAMES RESEARCH

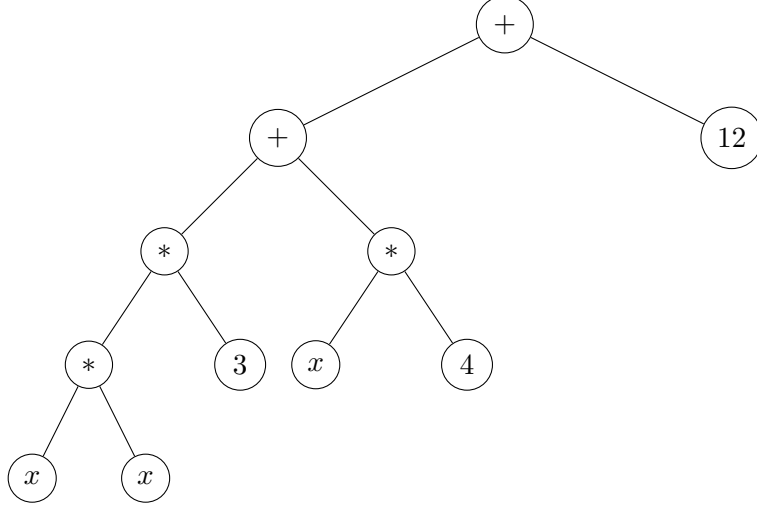


Figure 4.4: A GP tree modelling the example $y = 3x^2 + 4x + 12$

nodes or complete subtrees between solutions. On the other hand the *mutation* may affect only the values of a single node or all nodes from a specific subtree.

4.4 Modelling and Generating Game Mechanics

This section presents previous approaches to capture and model behaviour of software and algorithms. Even though our focus lies on game related approaches, we also briefly present approaches which are directly targeted at games.

4.4.1 Unified Modelling Language (UML)

The most versatile approach to represent algorithmic behaviour can probably be found in the Unified Modelling Language (UML). UML is a standardised language to express the structure and the behaviour of software or processes in general. Since its first proposal in the early 1990s it has become an accepted standard for software specification and inter-team communication within the software industry and the field of software engineering research. It features a variety of ways to visualise the architecture of a program, categorised into different diagram schemas. Two of the most well known diagrams are class diagrams, visualising the class structure of an object-oriented program, and a use-case diagram and sequence diagram to visualise the interaction of different components within- or users and other actors with the software. Furthermore, UML diagrams are often used with *code generation*. Code generation in this context means

that a software takes a user generated diagram and transforms it into source code. Such a system makes use of a template library, and substitutes - similar to a compiler - pre-defined tokens with the user defined content.

4.4.2 Stanford GDL

The probably most well known and versatile Game Description Language (GDL) was proposed by Love et al. at the University of Stanford and thereby is often referred to as *Stanford GDL*. The aim is to model all games that are discrete and have no hidden information as a game element, but was recently extended to include non-deterministic elements and partial information (171). GDL is a variant of Datalog, a subset of Prolog often used as query language for deductive databases. The rules of a game are given as Datalog rules. Games are modelled as state machines and players' inputs act as transitions. The outcome of the game (winning, losing etc.) is determined by which end state is reached. Unfortunately, such explicit representations are impractical with more complex games, e.g. chess has approximately 10^{31} different states.

Instead, in GDL each game consist of a database schema, and each gamestate corresponds to an instance of this schema. The database holds the states of all objects and variables of the game in the current state. For example, a gamestate of Tic-Tac-Toe is listed¹ as:

- cell(1,1,X)
- cell(1,2,B)
- cell(1,3,B)
- cell(2,1,B)
- cell(2,2,B)
- cell(2,3,B)
- cell(3,1,B)
- cell(3,2,B)
- cell(3,3,B)

The game's actual logic is expressed in logic statements rather than tables (an explicit state transition function). The key elements of the GDL vocabulary are the following words:

- role(a) means that a is a role / player in the game.
- init(p) means that the datum p is true in the initial state.
- true(p) means that the datum p is true in the current state.

¹The example and details are taken from GDL's official webpage: <http://games.stanford.edu/language/language.html>

4. RELATED COMPUTATIONAL INTELLIGENCE IN GAMES RESEARCH

- $\text{does}(r,a)$ means that player r performs action a in the current state.
- $\text{next}(p)$ means that the datum p is true in the next state.
- $\text{legal}(r,a)$ means it is legal for r to play a in the current state.
- $\text{goal}(r)$ means that player r 's goal is achieved in the current state.
- terminal means that the current state is a terminal state.

The actual game mechanics are split into conditions (the *legal* keyword) and consequences (the *next* keyword); though *next* rules are not directly bound to *legal* clauses what allows them to be triggered also implicitly by players' actions.

However, GDL is much more verbose than Browne's Ludi (see section 4.4.4). What Browne abbreviates as "in-a-row 3" is expressed in GDL as:

- $\text{line}(P) \text{ } i = \text{row}(M,P)$
- $\text{line}(P) \text{ } i = \text{column}(M,P)$
- $\text{line}(P) \text{ } i = \text{diagonal}(P)$
- $\text{row}(M,P) \text{ } i = \text{true}(\text{cell}(M,1,P)) \ \& \ \text{true}(\text{cell}(M,2,P)) \ \& \ \text{true}(\text{cell}(M,3,P))$
- $\text{column}(M,P) \text{ } i = \text{true}(\text{cell}(1,N,P)) \ \& \ \text{true}(\text{cell}(2,N,P)) \ \& \ \text{true}(\text{cell}(3,N,P))$
- $\text{diagonal}(P) \text{ } i = \text{true}(\text{cell}(1,1,P)) \ \& \ \text{true}(\text{cell}(2,2,P)) \ \& \ \text{true}(\text{cell}(3,3,P))$
- $\text{diagonal}(P) \text{ } i = \text{true}(\text{cell}(1,3,P)) \ \& \ \text{true}(\text{cell}(2,2,P)) \ \& \ \text{true}(\text{cell}(3,1,P))$

The actual definition of Tic-Tac-Toe is given in the language description document (172) and spans over a few pages. GDL is also used (and was designed for) in a "general game-playing competition". There, artificial players try to compete over winning previously unseen games modelled in GDL. Ultimately, no approaches to evolve game mechanics using GDL are known to the authors.

4.4.3 Answer Set Programming

Another approach to model game mechanics was published by Smith et al. His game engine "Ludocore" (173) is intended as a prototyping tool for game developers. It is based on answer set programming (ASP), and provides an API for common functions such as game states and game events. ASP is a subset of Prolog, and programs are given in the form of logical statements:

- $\text{wet} \text{ :- raining.}$
- $\text{wet} \text{ :- sprinkler_on.}$
- $\text{dry} \text{ :- not wet.}$
- :- not wet.

The example translated into plain English defines, that the ground is wet if it's raining and/or the sprinkler is on. It further defines "dry" as "not wet", and that "wet" is true (technically it defines that "wet" is not unprovable). The example above is

actually taken from another of Smith’s publications, using ASP for procedural content generation of game rules. His project “Variations Forever” (174) uses ASP to search a generative space of mini-games which all take place on a rectangular grid. The generator, based on an ASP solver, populates the grid with different coloured agents, assigning each colour a movement model. Furthermore, each agent type is assigned a table of effects in case they collide with other agents (plus a player controlled agent) or obstacles which are also placed in the game world. Possible effects are “bouncing the agent back”, “kill the agent”, or similar. Analogously a goal of the game is created. Although slightly more complex, this work is similar to the experiment done by Togelius and Schmidhuber, presented in section 4.4.6.

As a concluding remark it seems notable, that a similar language to ASP has also been used in commercial game creation software. The software “Zillions of Games” (175) has an active community of game designers at the time of writing.

4.4.4 Ludi

The work by Cameron Browne mentioned in section 3.2.4 is part of his “Ludi” framework (95). Ludi is a framework to define and evolve “combinatorial games”. Browne defines *combinatorial* as follows, covering a large subset of board games:

- **Finite** Games produce a well-defined outcome
- **Discrete** Turn-based
- **Deterministic** Chance plays no part
- **Perfect Information** Nothing is hidden from the players
- **Two-player**

The core idea of the Ludi Game Description Language are *Ludemes*, units of independently transferable game information (“game memes”), e.g. (*tiling square*) for a board layout or (*size 3 3*) for the game board’s size. Compared to Stanford’s GDL from section 4.4.2 the description of Tic-Tac-Toe in Ludi is rather brief:

```
(game Tic-Tac-Toe
  (players White Black)
  (board
    (tiling square i-nbors)
    (size 3 3)
  )
  (end (All win (in-a-row 3)))
)
```

4. RELATED COMPUTATIONAL INTELLIGENCE IN GAMES RESEARCH

The game consists of a definition of a board, square tiles and 3x3 in size, where players *Black* and *White* have both the victory condition to achieve three in a row.

Both Ludi and SGDL use tree structures to represent game mechanics. But compared to SGDL, Ludi is more brief and ludemes are more high-level compared to SGDL tree nodes, i.e. ludemes such as “three in a row” would require a whole subtree. On the other hand, if the victory condition should be changed to “one in each corner”, a new ludeme has to be programmed as the language does not permit a finer grained modelling of ludemes. However, the language as published allowed Browne to develop a framework which was able to evolve combinatorial games using the fitness measures presented in section 3.2.4. Like similar systems, Browne uses artificial agents (using standard Alpha-Beta search) to play the modelled games. As a state evaluator function he uses an ensemble of twenty different advisor functions which are fed into a linear combination which is configured for each game using an evolutionary strategy. Notable here is that the weight vector for the fitness weights was optimised using human play testing, and the initial population is filled using manually designed games. New individuals were created using probabilistic selection based on a individuals fitness and subtree crossover. Overall, the system was so successful in creating appealing games that Browne managed to have one of his games published as a physical board game. His game Yavalath (176) can be bought in toy stores or online.

4.4.5 ANGELINA

The ANGELINA¹ system is a project developed by Michael Cook (177). Cook intends to model and evolve arcade games: games where players run through simple two-dimensional worlds, try or not try to avoid objects, and perform simple quests such as finding keys to the next section of the game. The ANGELINA system performs individual evolution, i.e. the three components of an arcade game, maps, rulesets (or powersets) and character layouts, are evolved independently and use separate fitness functions through co-operative co-evolution. Maps are represented as two-dimensional integer arrays, that partition the game world into 8x8 areas and define textures and collision properties (solid or non-solid). Character layouts define the classes of enemies a player may encounter. Results presented by Cook in 2012 (178) show that the system is able to generate simple *Metroidvania* platform games, but indicate a gap between the fitness values and the players’ preferences. Furthermore, study participants had difficulties distinguishing between decisions the artificial designers made and implications made by the designers of the framework. To address the gap between fitness value and

¹a recursive acronym: A Novel Game-Evolving Labrat I’ve Named ANGELINA

enjoyment, Cook proposes the addition of other aspects into the fitness functions such as difficulty or simulated combat. Furthermore, Cook reports that one problem of the evolutionary system is that it converges on local optima.

Although ANGELINA has another genre of games in focus and uses a different representation than a tree of any kind, future work with SGDL could adapt the idea of parallel evolution. An interesting direction of research would be for example the independent evolution of combat and non-combat units. As we will see in section 8.1, it is hard to combine both types under the same definition of “balance”. Separating both unit types into different genomes could possibly overcome this problem.

4.4.6 Fixed length genome

Not all published approaches to game mechanics generation provide full description languages or grammars to represent game mechanics. Often a problem tailored representation of a specific game’s logic is sufficient. For example, Togelius and Schmidhuber (104) used a fixed length genome, an array of integers, to encode the outcome of a limited numbers of events in a “predator prey” game. In a Pac-Man-like game the player controls an agent in a two dimensional labyrinth. The labyrinth also hold a multitude of other objects which are controlled by artificial agents.

They defined a number of types of objects which could be in the game, a player avatar and three types of differently coloured game pieces. The genotype includes a behavioural model for each type (except the player agent), e.g. random moves or circular movements, and a matrix which defines the outcome of two game pieces colliding in the game. The framework contains several possibilities such as “teleport to random location”, “remove object A from the game”, “increase/decrease the player’s score”, or “the game ends”. The fitness function used in that experiment was described in section 3.2.5. Togelius and Schmidhuber themselves describe their experiment as a proof-of-concept to demonstrate the use of their fitness functions, and focussed less on the representation of the game rules; which is very basic as they report. A more critical point is, as they point out, that their agent did not show satisfactory performance in learning the games and would probably perform even worse with actually complex games.

4.5 Summary

This chapter presented a series of related approaches from the field of computational intelligence in games and machine learning. We focussed on previously published ap-

4. RELATED COMPUTATIONAL INTELLIGENCE IN GAMES RESEARCH

proaches to model and generate different aspects of games. We discussed the general approach of procedural content generation in games, also in regards of the question if computers or their work may be seen as “creative”. Although we do not intend to cover the aspect of computational creativity in detail, it should be mentioned that SGDL resp. the SGDL framework is partly designed to augment a human game designer’s creativity and serve as a rapid prototyping framework. It could be argued, that our evolutionary algorithms are artificial game designers, who either extend the work of a human, or create their own, and therefore could be called “creative”.

A significant part of this chapter presented techniques from the field of machine learning which form a basis for the experiments presented in the course of this thesis. Some of them will be used in our work, e.g. a cellular automaton is used in our case study for a map generator, described in section 6.2. We concluded this chapter with an overview of other approaches to model and generate game mechanics which influenced the development of SGDL. The next chapter will now shift the focus to the main contribution of this thesis, the Strategy Games Description Language (SGDL).

Chapter 5

The Strategy Games Description Language (SGDL)

This chapter presents the Strategy Games Description Language and its framework. Chapter 6 will present examples how it has been used so far. The language was developed and extended as necessary. At the time of writing, the modelling of commercially available strategy games was still an ongoing goal, and further additions to the language might be necessary. However, it was sufficient to perform the research presented in chapters 8 and 10.

5.1 Design Paradigms

The foundation of most modern linguistic research is Ferdinand de Saussure’s *structuralist* approach which defines “language” as a system of signs and rules how signs are arranged and may interact (grammar) so humans can communicate (transport meaning; semiotics) about objects which are not immediately present (179). While Saussure’s work is primarily about natural languages, his observations can be extended to engineered languages used in machine learning and artificial intelligence research. Although we do not intend to present a formal language capable of logical reasoning, we would like to discuss a few characteristics often applied to knowledge representation in regard of SGDL. These requirements were defined in an early stage of the SGDL project, and lead to the structure as it will be presented in the following sections.

Completeness SGDL is intended to be able to model all aspects of strategy games, including some published and successful games. This however is highly theoretical: as

5. THE STRATEGY GAMES DESCRIPTION LANGUAGE (SGDL)

we discussed in chapter 2. The boundaries that can be considered a strategy game - and what not - are blurry. Some cases can be clearly discarded from our point of view, e.g. first person shooters such as *Quake*, others like *Sim City* are debatable. Although some topologies of games have been proposed (by Aarseth et al. (180), among others) in the past, we found no concluding definition. We will therefore work with the definition presented at the end of chapter 2.

Human readable The structure of SGDL is representable in many forms, e.g. graphical trees or grammars. It can be also expressed in an XML document. This enables people to read and edit SGDL models manually at least on a micro level. In SGDL operations are modelled down to single operators. This naturally leads to a large number of nodes per tree. As a result, SGDL models might be hard to comprehend for humans on a macro level. We will discuss this aspect later in section 11.3.1.

Evolvable The space of games that can be expressed should be easily traversed i.e. be searchable. One of the main implications of this property is that the language should have a high locality, meaning that similar descriptions (genotypes) in general give rise to games (phenotypes) with similar fitness values. We defined SGDL as a tree structure so existing research and techniques from the field of genetic programming can be applied.

Locality Changing a single node in a SGDL tree should have limited implication on the gameplay the tree implies. The change of a whole subtree however should pose a larger change.

Concurrent turns Although not a property of the language per se, the concept of *concurrent turns* (used synonymously with *simultaneously* hereafter) affect the implemented framework. Many strategy games, e.g. *Civilization* or *Diplomacy*, permit simultaneously turns in some modes. Moves in turn-based strategies often require extensive planning and are time consuming, therefore many multiplayer strategy games offer the possibility of simultaneous turns. This aspect of strategy became more prominent with the introduction of computers as game-masters and network technology as discussed in chapter 2.

We observed that simultaneous moves in turn-based strategy games are merely used for convenience, i.e. to reduce the waiting time for players with minimal implications on the game rules. We therefore decided to keep the SGDL framework simple and only allow games with consecutive turns. Although the SGDL framework currently

includes no network component, multiplayer games at one screen or games against artificial players are possible. The implications on game mechanics are therefore the same. The subsequent problem for our “balancing” (see section 8.1) fitness is therefore the advantage of the opening move: when among two equally skilled players with equal starting positions and conditions one player has the advantage by just making the first move.

5.2 Basic Concepts and Terms

Strategy games, as we presented in section 2, have in common that the game world consists of a map which is populated by various objects. Those objects can either be static passive objects, e.g. debris, or dynamic objects players can interact with, and thereby influence the gameplay. All objects in the game world are unified under the concept of classes of *WorldObjects*.

WorldObjects are all the objects that are game-relevant. Some WorldObjects are in a spatial relation, i.e. directly on the map, others are abstract objects such as players or game mechanics that are invisible to the player. The first example could be a tree standing on the battlefield, while the second example could be a factory off the map; providing the player with reinforcements. Borrowed from object oriented software programming, every WorldObject is an instance of a class (*ObjectClass* in the following), defining a data state and a behavioural model. We will refer to the data state of a WorldObject as **Attributes** and its behavioural model as a set of **Actions**.

ObjectClass An ObjectClass is a template which includes business logic how instances may interact in the game. The term *business logic* is often used in software engineering to describe the actual behaviour of the software, excluding framework routines which handle input- and output-operations. Following the object oriented programming nomenclature, the attributes correspond to *member variables* while each action could be seen as a *method* that could be called on an object. The key difference is that the behaviour of actions resides in a dynamic model while programmed methods normally exist only as machine code during runtime in conventional programming languages. The method we use to model the behaviour of actions is a tree structure, which is also very common technique in genetic programming.

- **Attributes** are a data vector that can be associated with a WorldObject

5. THE STRATEGY GAMES DESCRIPTION LANGUAGE (SGDL)

- **Actions** are the abilities a *WorldObject* could enact. All instances of a certain *ObjectClass* share the same actions, but have their own copy of their *Attributes*.

Note: *WorldObject* and *ObjectClass* should not be confused with the terms *class* and *instance* of the language in which the framework is implemented (Java in our case). An actual object in the game is an instance of the (Java) class *WorldObject* which is associated with an instance of the (Java) class *ObjectClass*. Defining another class-object layer on top of an existing object oriented language enables us to be as independent from a certain implementation as possible.

Attributes An *Attribute* in general is a simple Object-Data relationship, i.e. a member variable of an instantiated object of a class. These immediate attributes will be called *Constant Attributes*, a simple data holder. The term “constant” here refers to the container itself, not the value. We will also present *Attributes* that require more context to determine their value, i.e. the location where the actual data resides is non-constant. A *WorldObject* always possesses at least one attribute, namely the *ObjectClass* it’s derived from. The most common attributes are x and y to position an object on the map.

Actions Actions correspond to methods from object oriented programming. Like a method body, the SGDL tree only specifies the static behaviour but not the control flow outside the method, i.e. an Action does not control when or if it is invoked. The control lies within the implementing system of a game engine. The engine controls whenever interactions of the player with the system may lead to action invocations or not (e.g. it may not be the player’s turn). SGDL is intended as a declarative language, although the Operator nodes in Consequence subtrees (describing assignments of variables) make it more a hybrid language between a declarative and an imperative approach.

SGDL is designed as a layer within a game engine with interface logic above. The invocation of an action is split into two parts: the **conditions** and **consequences**. Both share the concept of a *context vector* which is passed into the action to determine both the result of the conditions and (if applicable) the consequences. The **context vector** contains all *WorldObjects* passed down from the interface layer, including the object the action is triggered on (the *acting* object) and the game state. Again, SGDL does not specify the control flow, or how players actually interact with the game objects. SGDL just assumes, that at one point the player’s interaction with the game interface results in an invocation of a *WorldObject*’s Action. Creating an efficient user interface based on arbitrary game rules is unfortunately outside of the current scope of



Figure 5.1: Selecting an Action in the SGDL Game Engine

our research, and we resorted of a very basic interaction model: we provide a popup menu with an entry for each Action a `WorldObject` might perform at that game state (figure 5.1).

5.3 The SGDL Tree

The SGDL tree is a strongly typed tree, i.e. the type of potential child nodes are determined by the parent node. Similar to a grammar, only syntactically coherent trees are permitted. However, no assumption about semantic correctness is imposed. SGDL consists of a variety of different node types. The root node is reserved for meta such as version and other debug information used during development. At the top level the tree defines `ObjectClasses`, the game state template, the player state template, and the winning conditions. Figure 5.2 shows a general overview: a set of *ObjectClasses*, the player and game state (described in section 5.4.2, and a list of winning conditions specifying how to win a game. Figure 5.3 shows a complete overview of all node types defined so far. They will be described over the following sections and include:

- **SGDL** is a meta node and combines all subtrees into a single model.
- **ObjectClasses** are `WorldObject` templates.
- **Operators** are value assignments.
- **Comparators** are logical operations.
- **Actions** Define `WorldObjects` abilities.
- **Attributes** are various nodes that point to a data value. These include:
 - Constant Attributes
 - Property Nodes
 - Special Functions which are nodes that incorporate certain domain knowledge, that would be hard or impossible to model (e.g. map related functions).
 - Object Nodes (references to `WorldObjects`)

5. THE STRATEGY GAMES DESCRIPTION LANGUAGE (SGDL)

- **ObjectList/-Filter** Allow the combination/manipulation of several WorldObjects at a time.

5.3.1 Attributes and Object References

The major group of node types are classified as *Attributes*. Although they work differently they all have in common that they pass a data value up to its parent node during a condition or consequence test. The simplest node, the *ConstantAttribute*, is a leaf node, that can be attached to an *ObjectClass*, and thereby become a mutable attribute, or to another node in the tree. The latter makes a *ConstantAttribute* an actual constant, as nodes within the tree can't be referenced or manipulated; the term *ConstantAttribute* just refers to a static reference to a variable otherwise. To refer to *ConstantAttributes* attached to *ObjectClasses*, i.e. properties of *WorldObjects*, a *Property Node* is used. That node defines the attribute name to be resolved from an adjacent *Object Node*, the game state, or the player states. Formally, an *Object Node* is an attribute that returns a *WorldObject* instance based on its configuration. The *Property Node* resolves the data value from that *WorldObject*.

Example 1: The most common case, seen in figure 5.4, is accessing a *WorldObject* from the context vector. Objects are referenced by their index in the vector, starting with zero for the acting object. The minimal example in figure 5.4 has a *Property Node* referring to the attribute “a” of its children *Object Node*. That node refers to *WorldObject 0*, which is the actor, i.e. the object a player used to click on. All node types combined express logical or mathematical expressions as a tree.

5.3.2 Conditions

A Condition is a sub tree, with a comparator as a root node, that returns a boolean value up to its parent. The boolean value “true” indicates that the condition may be fulfilled. It consists of a symbol representing a logical comparison, and one or two operands. The inputs of the operands - and therefore the child trees - are ordered, i.e. asymmetrical operations are well defined ($3 < 5 \neq 5 < 3$), left and right subtree of a node correspond to the left and right side of an equation. Furthermore, the data types must match the comparator, e.g. determining the lesser or equal string literal of two may not be defined, but testing if they are equal is. The following boolean operations are defined:

- The arithmetical operations $<, \leq, \geq, >$ for any numerical values

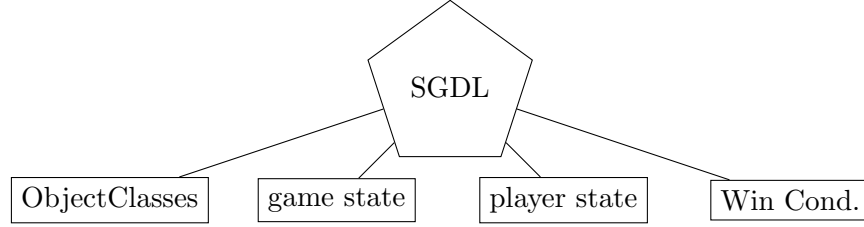


Figure 5.2: The overall SGDL tree.

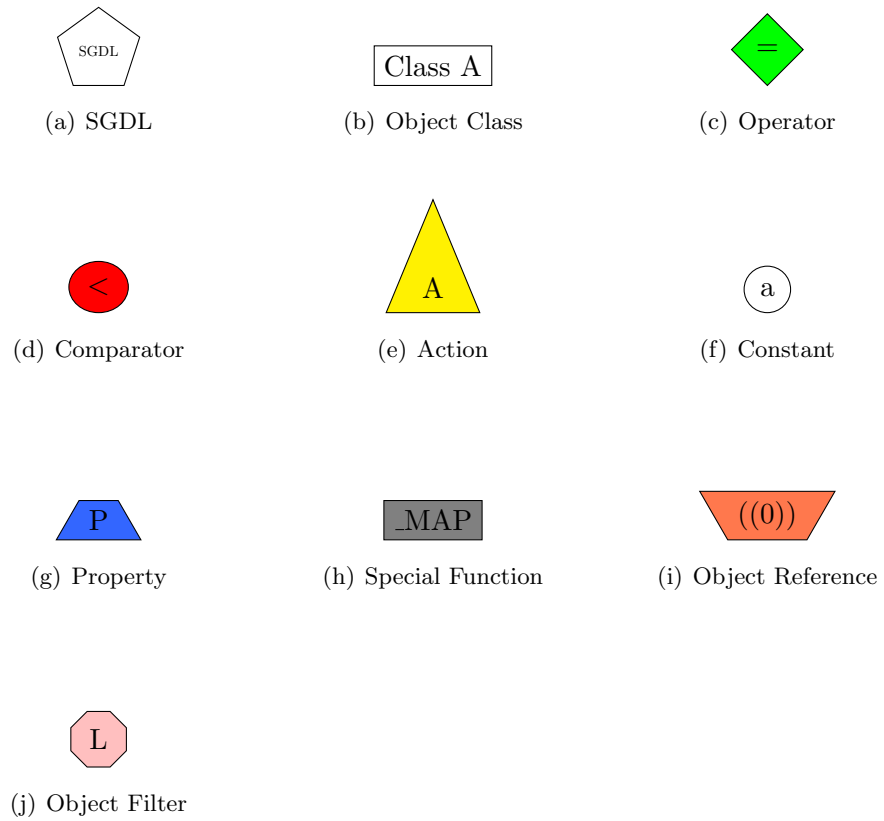


Figure 5.3: Overview and graphical representation of the SGDL node types.

5. THE STRATEGY GAMES DESCRIPTION LANGUAGE (SGDL)

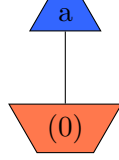


Figure 5.4: Accessing a property

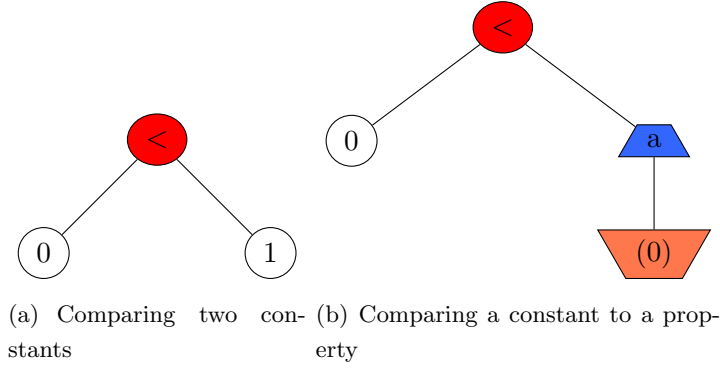
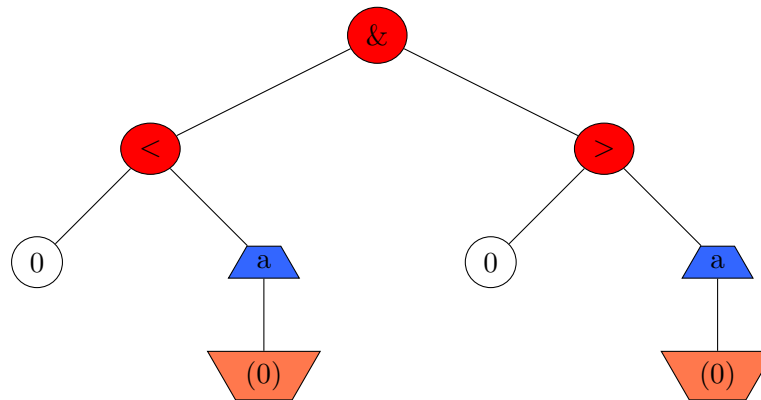


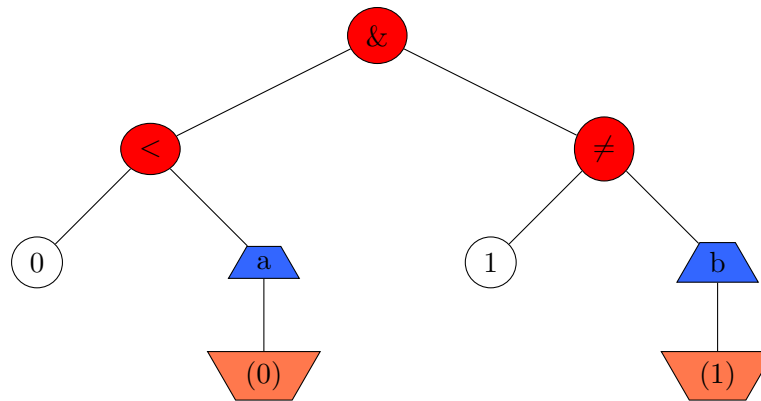
Figure 5.5: Two elementary examples of subtrees comparing constants.

- The logical operations *and*, *or*, *nor*, *xor*, *not* for boolean values
- The operations $=$, \neq for every data type

The smallest example is the comparison of two *Constant Attributes*, which would always return the same logical value (e.g. the example in figure 5.5(a) would always be true). A more relevant examples would be comparing an attribute to a constant. The example in figure 5.5(b) displays a subtree which returns *true* if the attribute “a” of the acting WorldObject is larger than zero. The output of a *Condition* can be fed in turn into another condition as an input, allowing the definition of cascading logical expressions. Suppose we extend the example above, as in figure 5.6(b). We can add a reference to a second object and a test to see if its “b” attribute is unequal to one. **Note:** The SGDL model only assures a correct syntax of the model. A further direction of research (we did not explore yet) would be the implementation of model checking techniques. Even though we partially address this issue in one of the experiments in section 9.2.1, the language allows the definition of oxymorons. E.g. the example seen in figure 5.6(a) would never return true.



(a) A Condition that would never be true.



(b) Several Conditions linked together

Figure 5.6: Extended conditions

5. THE STRATEGY GAMES DESCRIPTION LANGUAGE (SGDL)

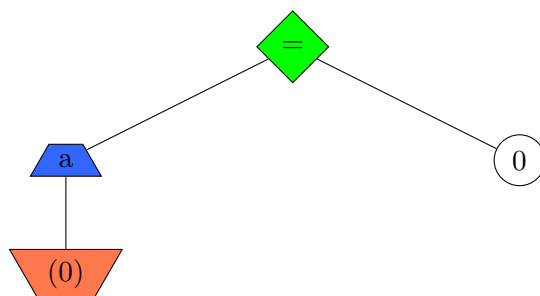


Figure 5.7: The attribute “a” of the acting WorldObject is assigned the constant value “0”. This implies that “a” has been defined as a numerical value before. Data types are not displayed within the visualisation of a tree.

5.3.3 Consequences

Consequences work analogous to conditions, but instead of passing a value in the vertical direction of the tree, they work horizontally from right to the left. The subtree’s root node is an *Operator node*, which evaluates its right subtree and assigns and/or combines it with the result of the left subtree. This requires that the result and the target are of compatible data types, i.e. a Consequence can’t assign a nominal value to a numeric Attribute or vice versa. It also requires that the left subtree can accept a data value to pass it down to its children. Analogous to resolving values, the common example is a *Property Node* that works as an assignment target, as seen in figure 5.7. This assigns the value 0 to the Attribute “a” of the acting object, assuming that “a” is a numeric attribute.

Creating WorldObjects Consequences can be used to create new WorldObjects on the map. A plain *Object Node* with the keyword **_NEW** can be used as a source for a new object. The node then takes an alphanumerical *Constant Attribute* with a class name as a child. To place a new WorldObject on the map a *SpecialNode* **_MAP** is used. This case is presented in section 5.4.1.

5.3.4 Actions

Actions are sub trees with an *Action node* as a root. They are always part of an ObjectClass and represent an ability a WorldObject may use within the game. They further posses several Conditions and Consequences as child nodes. While theoretically possible to exist as a stand-alone node, an Action always has at least one Consequence. If no Conditions exist, the Action is treated as unconditional. Whenever an *Action* of

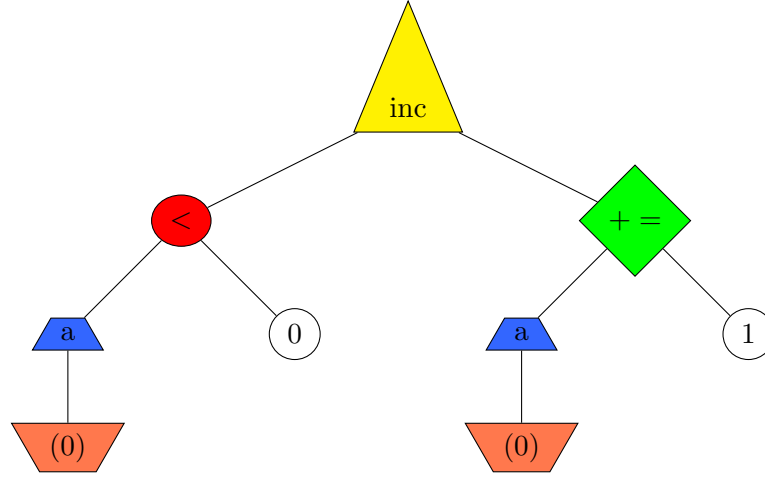


Figure 5.8: The action “inc” has one Condition, the property “a” of the acting object must be lesser than zero, and one Consequence. The $+ =$ operator combines the value of the right subtree (here a constant) and combines it with the left subtree. Operators that need two operands evaluate the result of the left subtree as well first, and assign the result back into the left subtree. The textual representation of the tree would be: $if (O_0.a < 0) then O_0.a + = 1$

an WorldObject is triggered during gameplay, all its conditions are evaluated. If at least one condition returns false, the action has no effect. If all conditions evaluate to “true”, the action’s *Consequence* nodes take effect. Consequence nodes are rolled out in a sequential order (while Conditions are unordered). A number adjacent to the edge (figure 5.10) designates the order of a Consequence in the sequence. If two or more Consequences have the same index, they come into effect random order resp. in the order the implementation stores them. If no index is given, the default 0 is assumed.

Example 2: the Action “inc” in figure 5.8 has one Condition to check if the Attribute “a” of the acting WorldObject is less than zero. If invoked, *inc* increments the attribute by one.

5.3.5 Multiple Consequences

Some effects require Consequences which are applied simultaneously. For instance, a diagonal move on a Chess board requires the alteration of both the x- and y- coordinate of a game piece. If the changes are rolled out purely sequential, the following problem occurs: Starting with the board configuration as seen in figure 5.9(a) the white player would like to use his bishop on H3 to beat the black pawn on E6, resulting in the state

5. THE STRATEGY GAMES DESCRIPTION LANGUAGE (SGDL)

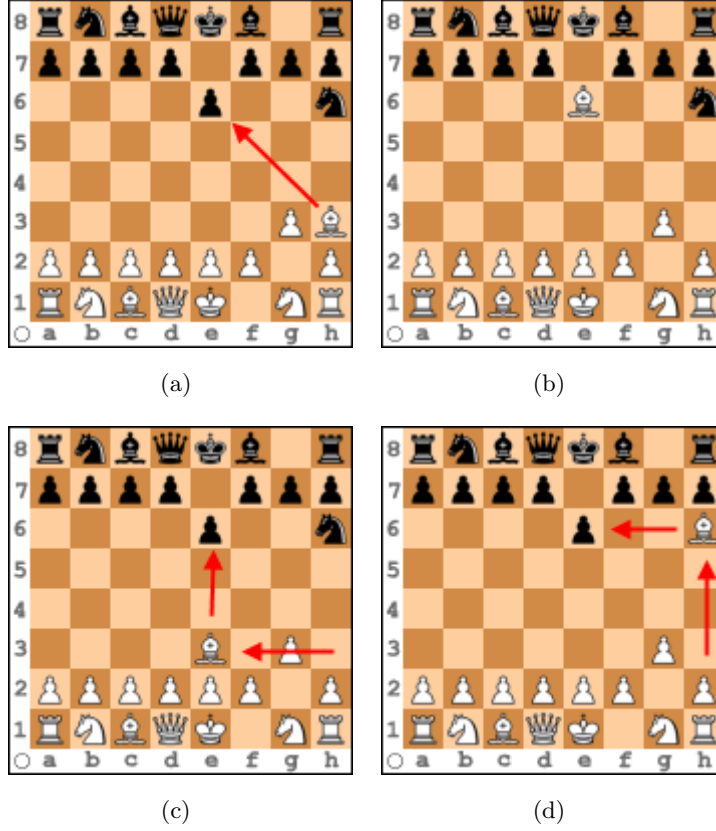


Figure 5.9: A horizontal move may cause unwanted side effects due to multiple attribute changes.

seen in in figure 5.9(b). If the change of the column would be applied first, and the movement between rows afterwards, the bishop virtually moves over tile E3 (as seen in figure 5.9(c). While in this configuration this has no effect, figure 5.9(d) illustrates what problem occurs if column- and row-assignments are switched: the bishop not only eliminates the pawn, but also eliminates the black knight in the intermediate step. To overcome this problem game designers may mark Consequences as simultaneous, but this does not solve the problem of concurrent assignments on the computing level. With a single computation unit, parallel executions are often modelled as sequential operations in arbitrary order. However, in the worst case the chess movement problem, as outlined above, may occur. Although this is a restriction imposed by the underlying implementation, and might not be necessary when simultaneous variable assignments can be made, e.g. π -calculus implementations (181). However, to be as independent as possible from the underlying implementation as possible, we approach this problem

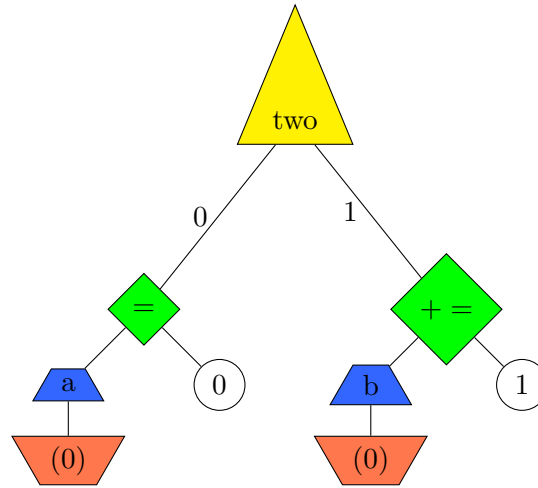


Figure 5.10: Ordered Consequences can be denominated with indexed vertices.

in SGDL by attaching indices to the outgoing vertices of an Action node. Each Consequence takes place in the order specified. The following example (figure 5.10) has two Consequences (but no Condition). After the Attribute “a” is assigned a value, it is also incremented. The indices at the vertices indicate the order they are supposed to be applied.

5.3.6 Action as Consequences

Actions themselves can be Consequences of other actions: if an Action is triggered as a Consequence, its Conditions are tested and then its Consequences are invoked before any further Consequences of the original Action take effect. A common example is the combination of an “attack” action, that subtracts health from a target, and a “kill” action, that eliminates the target from the game if its health is below zero. The graph for this example can be seen in figure 5.11.

Note: the special keyword **_SELF** is actually not an attribute name but refers to the WorldObject itself. Setting it to *null* (eliminating it) will delete the object from the game (and therefore the map) after the Condition/Consequence evaluation is done.

5.3.7 Object Lists and Object Filter

The last node type concerns the combination and manipulation of multiple WorldObjects at a time. ObjectList nodes combine several Attributes, that would return a WorldObject to their parent, into one single node. Additionally, ObjectLists can have additional Condition nodes as children which control which WorldObjects will be re-

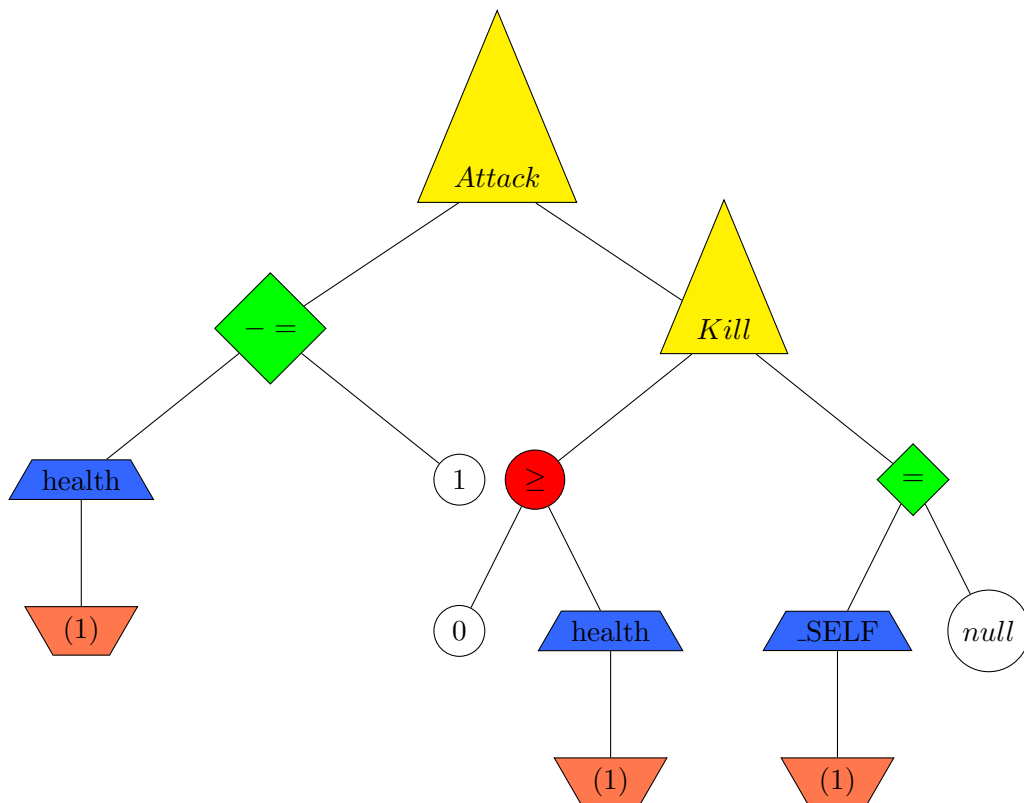


Figure 5.11: Actions can be Consequences of other Actions. Their Consequences will be only applied if the additional Conditions are met. The tree's function can be described as: "If a unit attacks a target, then decrease the target's health by one and check if the target has more than zero health points left. If that is not the case, then remove the target from the game (assign *null* to it) and therefore kill it."

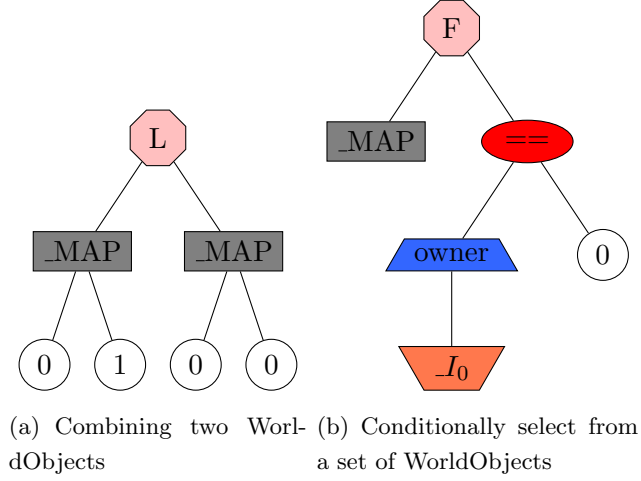


Figure 5.12: Examples for using an ObjectList.

turned to the ObjectList’s parent, and therefore act as a filter. Figure 5.12 shows examples of both node usages. Inside the subtree of an ObjectList a special identifier I_0 can be used for the object tested by the Objectlist node. Cascaded ObjectLists provide additional indices I_1, I_2, \dots, I_n .

5.3.8 Winning Conditions

With the all node concepts established, we can specify the final component, the definition when a game ends. Attached to the SGDL root node are a number of Conditions, that are checked after every player’s turn. While normally Conditions are tested against WorldObjects on the map, winning conditions’ context vectors are filled with a player’s state. Conditions may refer to this directly or other WorldObjects on the map via a *Special Function* node **_MAP** to determine if that player has won the game. If a winning condition returns true, that player has one the game. Each winning condition is tested against all players. Winning condition for specific players may be defined by explicitly referencing their player id. Figure 5.13 shows two examples for common winning conditions.

5.4 Surrounding framework

The game mechanics of a strategy game never exist on their own. So far we have only described how SGDL (sub-)trees interact with WorldObject instances that are passed into Conditions and Consequences. To provide additional structure necessary

5. THE STRATEGY GAMES DESCRIPTION LANGUAGE (SGDL)

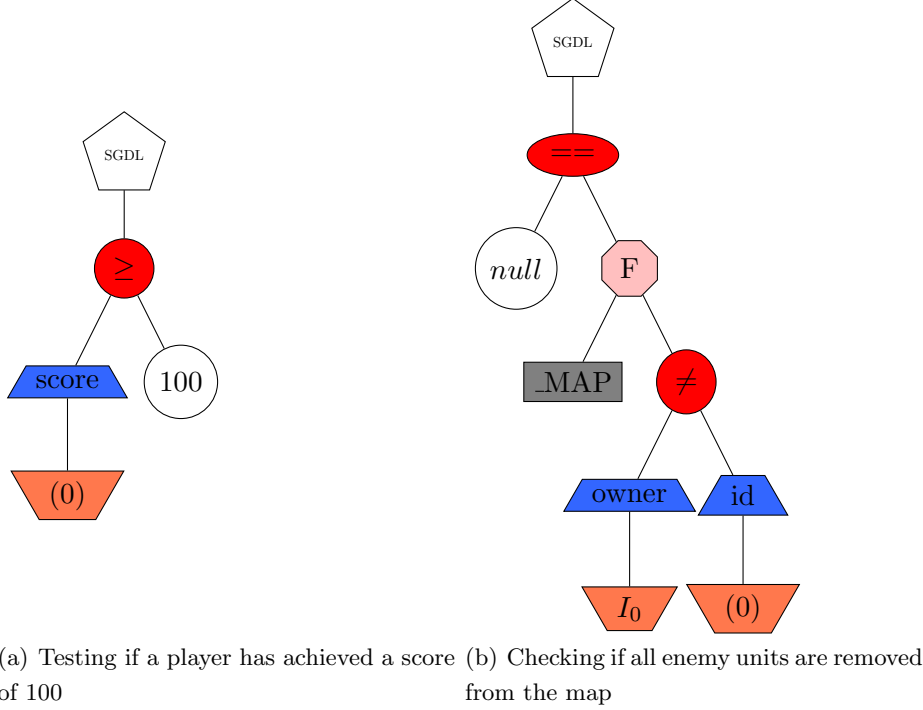


Figure 5.13: Two examples of winning conditions

for complex strategy games, the language also specifies several surrounding concepts that are common in modern computer games.

5.4.1 Maps

Maps are the central game element of strategy games. Their purpose is to define a spatial relation between units and other objects. Game maps are normally a two (sometimes three-) dimensional data structure and are further classified as either continuous or tile based. Continuous maps are mostly used in real-time strategy games and each object is located using real values. Tile based maps are more abstract and divide their space into separate spaces, often in the form of squares or hexagons, and sometimes in complex forms, e.g. continents in *Risk*. The current version of the SGDL game engine only considers tile-based maps as it is primarily intended for turn-based games. However, the language does not make any assumptions about the shape or type of the map and this is handled in the implementation layer. What can be specified, when referencing a WorldObject on the map, are coordinates. The most common are x-coordinates and y-coordinates, but a hierarchical ordering in continents and coun-

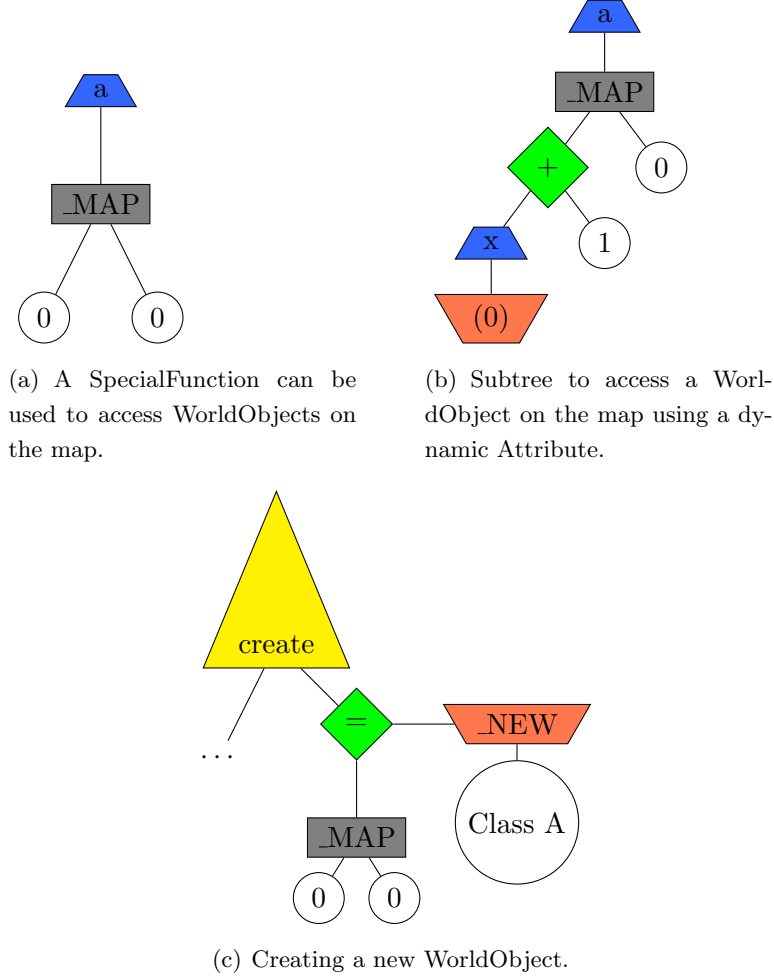


Figure 5.14: Three ways of accessing a *WorldObject* on a map.

tries might also be possible. To access a certain *WorldObject* on the map we can use a *SpecialFunction* node and let it act as a **MAP**¹ function. Two *ConstantAttributes* (or any other numeric *Attribute*) are used to provide the coordinates. Together with a *Property Node* the tree shown in figure 5.14(a) references the attribute “a” of the *WorldObject* at 0,0.

We can easily replace the *Constant Attributes* with something more complex. Assuming that the map is rectangular, we can address the object east (i.e. adding one to its x-coordinate) of the acting object as seen in figure 5.14(b). The same *Special Function* can be used to place *WorldObjects* on the map if used as an assignment target.

¹The “_” denotes a system function. Future versions might also include user specified routines.

5. THE STRATEGY GAMES DESCRIPTION LANGUAGE (SGDL)

Together with an *Object Node* set to **_NEW**. The example in figure 5.14(c) creates a new *WorldObject* of *Object Class* “Class A” at map tile (0,0).

Distances To determine the spatial distance between two tiles, SGDL provides a second *SpecialFunction* **_DISTANCE** which refers to another function of the underlying implementation. For instance, this could be the Euclidean distance or any other distance function applicable to the map. The **_DISTANCE** node takes two *WorldObjects* as inputs and provides a numerical distance. The default implementation assumes x- and y-coordinates and calculates the Euclidean distance. The relation between two *WorldObjects* from the Context Vector can be determined as seen in figure 5.15(a). This only works if two *WorldObjects* are involved in an *Action*. To simply calculating the distance between two tiles or a *WorldObject*, SGDL provides a third function **_MAP_TILE** which references the tile itself at a given position. Therefore it is also possible to calculate the distance of between two tiles, both unoccupied by *WorldObjects*. The current implementation to **_MAP_TILE** is limited to two input nodes, i.e. it does not support calculating the length of a path. On a rectangular grid map, the example shown in figure 5.15(b) would evaluate to “1”.

Tile properties Like any other *WorldObject*, a map tile may have properties. This is especially useful for modelling resources or other concepts that affect map tiles: e.g. move actions could be slowed on swamp tiles. Like *WorldObjects*, properties of map tiles can be accessed using a *PropertyNode*, but instead of using an *ObjectNode* as a child, a *SpecialFunction* **_MAP_TILE** is used. Tiles are then treated like any other *WorldObject*.

5.4.2 Game- and Player State

Like any other games, strategy games use variables that are not directly represented on the map but form a context of gameplay. Hidden or abstract information is normally stored in a set of global variables or auxiliary data structures. Examples range from time of the day, to the weather conditions, score counters, and timers. We refer to this as the “game state”, differentiating it from the objects that are on the map. We further segment this into game- and player variables, where the latter hold values that are only applicable for a certain player. So each player has his own score variable or a set of resources he can dispose.

The SGDL framework provides the concept of *game state* and *player state*. Both are a collection of *Constant Attributes* which designers can specify in the appropriate

5.5 Comparison to other Game Description Languages

node as seen in figure 5.2. The SGDL framework gives game designers the opportunity to access these values like any other *WorldObject*. The game state will be created and set to its default value upon game start, and can be accessed through a *Property Node* without an *Object Node*. Additionally, the property name has **_GAME** as a prefix, as seen in figure 5.16.

Player State Although it would be possible to organise game variables for several players simply using hierarchical names (e.g. `player1.money` and `player2.money`), we specified also a “player state”. For each player, a copy of the player state template will be made at game start. *Property Nodes* can be used to access a player’s state. An additional numerical *Constant Attribute* can be used to determine a specific player. If that is omitted, the current player’s state will be used.

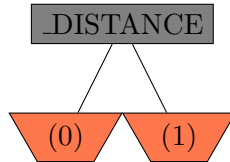
5.5 Comparison to other Game Description Languages

After introducing all elements of SGDL, we would like to make a brief comparison between a few of the previously published game description language, namely Ludi (4.4.4), Togelius and Schmidhuber’s fixed length genome (4.4.6), Angelina (4.4.5), and answer set programming (ASP, 4.4.3). We would like to address a few paradigms as discussed in section 5.1: range of games, verbosity, human-readability, syntax, and level of detail.

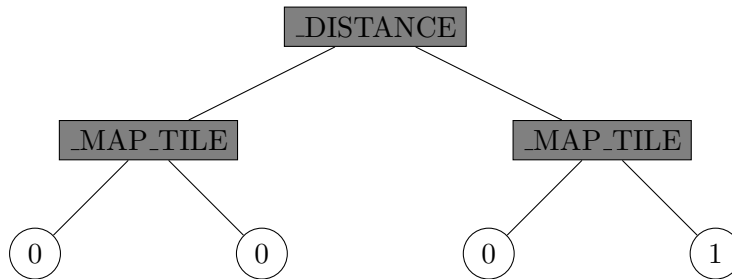
Ludi Ludi covers clearly a related range of games as SGDL games. As Ludi intends also to express classic games such as Chess or Checkers, there is clearly an overlap with SGDL. The minimal atom in Ludi (a “ludeme”) seems to encapsulate more behaviour than a node in a SGDL graph. This makes a game definition in Ludi clearly more human-readable, especially with complexer programs. It lowers also the verbosity of the language. This however has impact on the locality of the language, as single ludemes may change the gameplay significantly. The system relies on a library of known ludemes. Additional behaviour must be coded, e.g. if the system knows a three-in-a-row ludeme, but the game designer would like to express three-in-a-dashed-line concept the language has to be extended. SGDL relies more on modelling behaviour on a lower level.

Fixed length genome The fixed length genome used by Togelius and Schmidhuber covers only a small range of games and has clearly the most abstract representation of all game description languages discussed here. This results in a very low human-readability and knowledge of the experiment to interpret genomes. Behaviour however is modelled on a very high level, giving the representation a very low verbosity.

5. THE STRATEGY GAMES DESCRIPTION LANGUAGE (SGDL)



(a) Determining the distance between two WorldObjects. The required attributes depend on the map used, i.e. a WorldObject on a rectangular grid map will need x- and y-attributes.



(b) Measuring the spatial distance between two WorldObjects.

Figure 5.15: Examples of using the `_DISTANCE` SpecialFunction

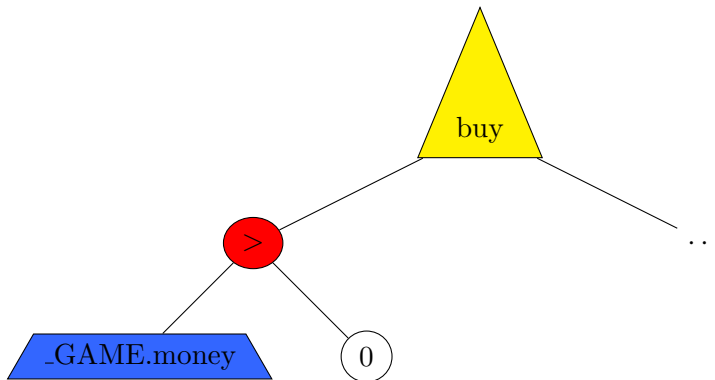


Figure 5.16: Accessing a game state variable in an Action. A Consequence will only be evaluated if the global variable “money” is greater than zero. Like any other Condition, SGDL trees imply that values are comparable. If “money” would be a string literal, our implementation would generate a runtime error.

Angelina Angelina addresses a completely different sets of games than SGDL. The arcade games relate more to the set of games expressed by the fixed length genome. However, Angelina uses an XML format to express its games which makes the genomes far more human-readable by trading off some verbosity. The level of detail is similar to Ludi, much higher than the fixed length genome, but also much lower than ASP or SGDL.

Answer Set Programming ASP is intended to express a similar set of games than the fixed length genome or Angelina. The main differences is that it builds on a full featured programming language (AnsProlog) which offers a high level of detail does to single operators. Even though SGDL does not build on an existing language, it reaches a similar degree of detail. The declarative programming differentiates it somewhat from the other approaches presented. This, depending on the designer, have some impact on the human-readability.

In comparison to the approaches presented above, SGDL covers a range of games which only Ludi relates to. However, a general comparison seems possible. SGDL is probably the most verbose language. This is due to its high level of detail, i.e. modelling behaviour on the level of single operators. The human-readability may depend on the designer's preferences, as some people might prefer a textual representation over a graphical one. This is especially true for complexer programs. We believe though, that a graphical representation is more intuitive.

5.6 Summary

This chapter introduced a domain specific language for expressing strategy games mechanics, the Strategy Games Description Language (SGDL). We discussed its design paradigms, introduced its basic concepts, and presented the different node types and how to combine them into simple game rules. Furthermore, we introduced the basic concepts of the surrounding framework and how it stores data regarding the game state. The chapter concluded with a brief comparison of SGDL to other Game Description languages. The next chapter will present how SGDL may be set in practice by demonstrating its use with a series of small example games.

5. THE STRATEGY GAMES DESCRIPTION LANGUAGE (SGDL)

Chapter 6

SGDL in Practice

This chapter presents how the SGDL framework was used to model various games. Some of the games were the basis for the experiments described in chapter 10. Furthermore, this chapter addresses another aspect of strategy games, map generation. A map generator for one of our games, modelled in SGDL, is presented in section 6.2.

6.1 Example Games

This section will present the games that have been realised with SGDL for our experiments. Each game builds on the previous one and increases in complexity. All games were modelled manually, but used for several evolutionary computing experiments to automatically determine the best values for various parameters and attributes. The following therefore just describes the structure of games without concrete parameter sets.

6.1.1 Simple Rock Paper Scissors

Simple Rock Paper Scissors (Simple RPS) is the smallest game realised, and simulates the battle between two units on an abstract level. The game is loosely inspired by the hand game “Rock Paper Scissors”. It features three different unit classes (“Class A”, “Class B”, and “Class C”) with identical abilities, namely attacking another unit. An attack simply reduces the enemy unit’s health by a strength value and costs one unit of ammunition. The strength against other units is denoted as “attack A” etc. The players take consecutive turns, and the player who loses his unit first loses the game. Each ObjectClass has five attributes:

- **ammo** is the number of shots that can be fired

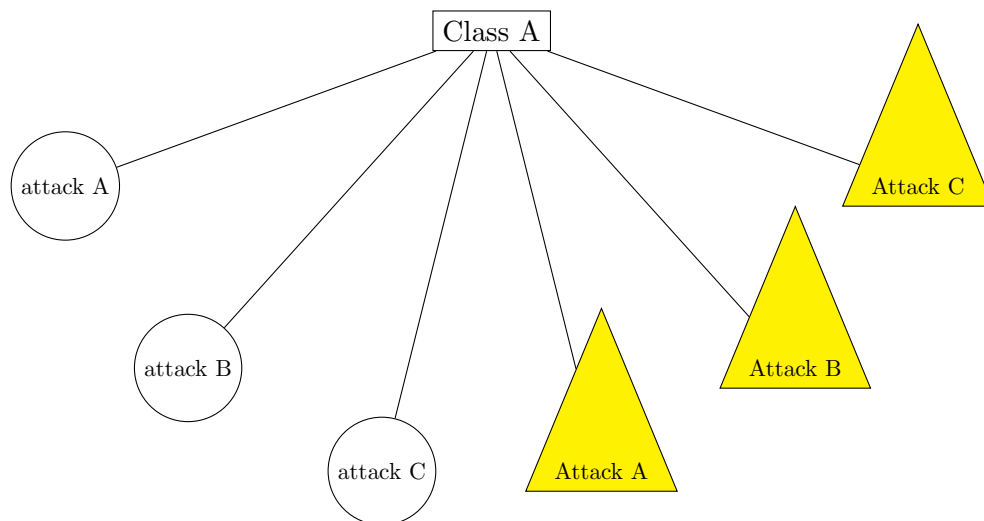


Figure 6.1: Subtree of “Class A” of “Simple Rock Paper Scissor”

- **health** is the number of damage a unit can take before it is removed from the game
- **attack A** denotes the attack strength against another WorldObject of Class A
- **attack B** [analogous for Class B]
- **attack C** [analogous for Class C]

The exemplary tree for Class A can be seen in figure 6.1. The “Attack A” action (and all attack actions analogously) can be seen in figure 6.2. It can only subtract health from a target object if the classes match, the target is not dead, and the acting object has ammo left. The amount of health subtracted is determined by the appropriate “attack” attribute. The winning condition was written in Java code and therefore not modelled in SGDL. This is simply an anachronism from an early stage of the development of SGDL, where winning conditions were not added yet.

6.1.2 Complex Rock Paper Scissor

Complex Rock Paper Scissor (Complex RPS) extends Simple RPS by a spatial component. A 10x10 rectangular grid map was introduced and instead of one unit, a player starts with three different units; each of one ObjectClass. Figure 6.3 shows an overview screenshot. The rules remain the same: players take consecutive turns and the player who loses all his units first loses the game. Additionally the concept of maximal and minimal shoot ranges was introduced.

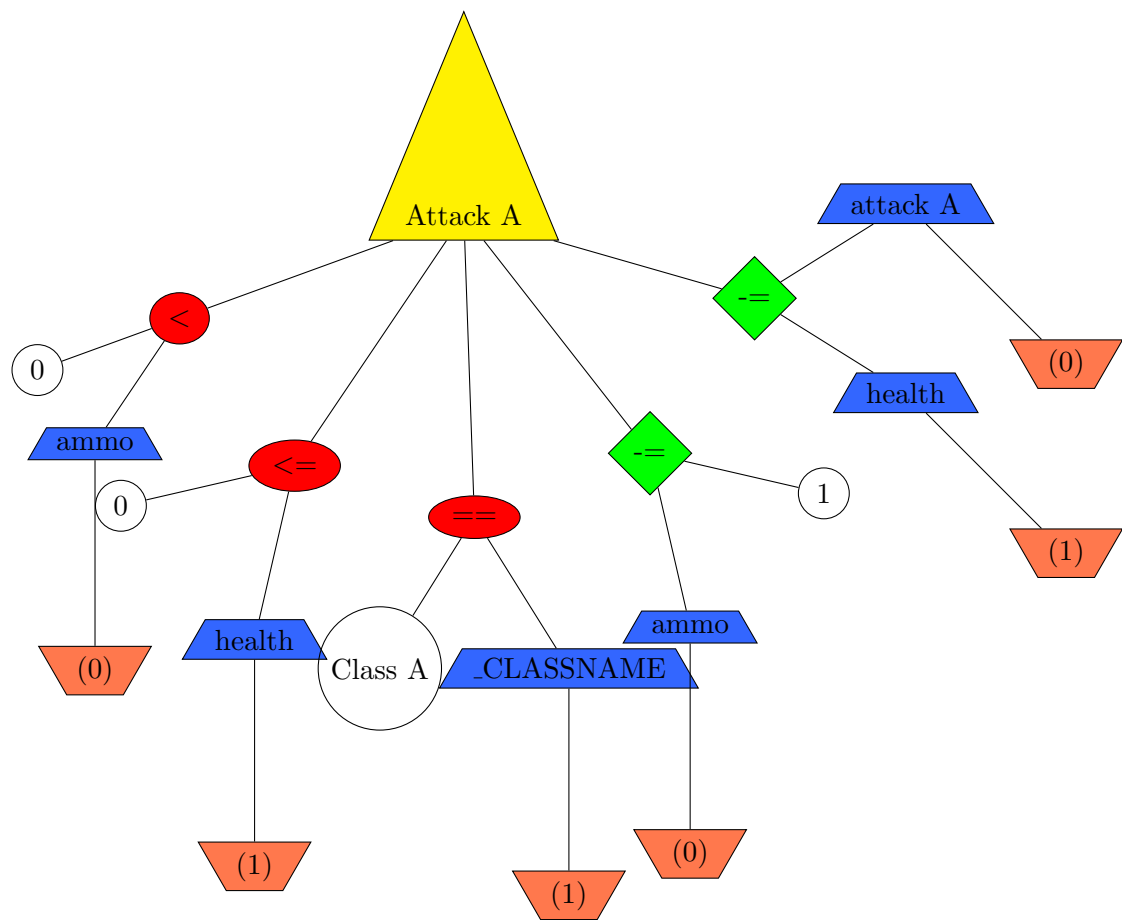


Figure 6.2: The attack action in simple Rock Paper Scissor. The child nodes model (from left to right): a) attacker has more than zero ammo b) the target has more than zero health c) the target's class equals a specific class (referenced by class name "Class A") d) subtract one from attacker's ammo e) subtract the attack value from the target's health.

6. SGDL IN PRACTICE

The attack actions as seen in figure 6.2 were extended with the condition checks as seen in figure 6.4 to model the range checks. At the same time the three ObjectClasses were extended with the following attributes:

- x - the horizontal coordinate
- y - the vertical coordinate
- minimum range for shooting
- maximum range for shooting

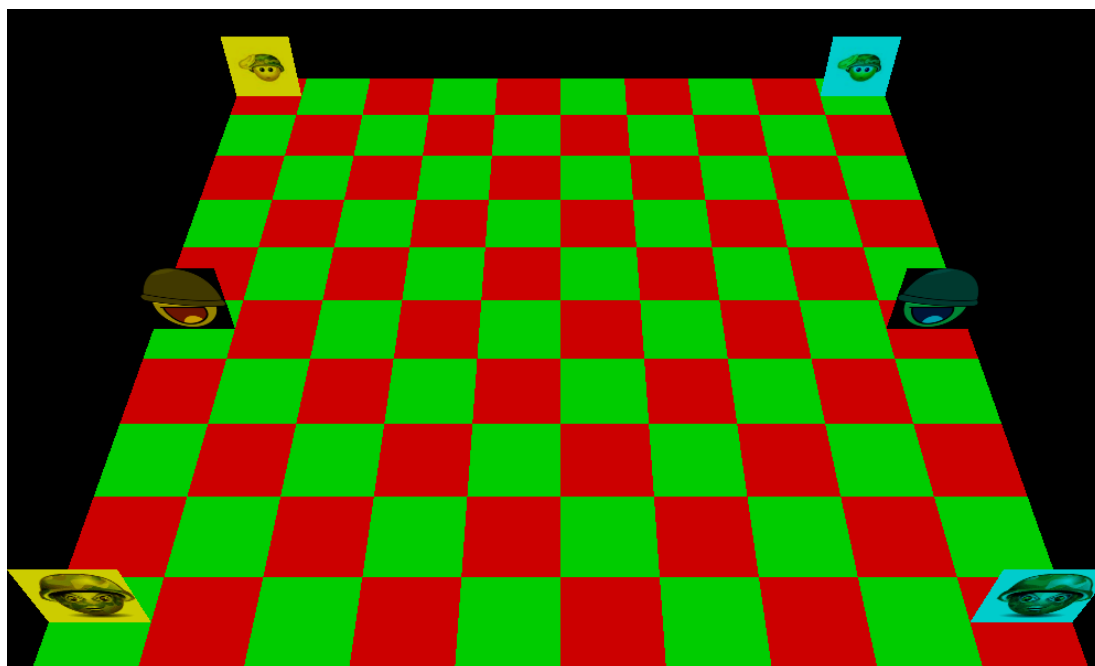


Figure 6.3: A development screenshot of “Complex Rock Paper Scissors” with placeholder graphics. One player starts on the right, the other player on the left. Each of the six tokens represents a unit. Units with the same image (but different colour) are of the same unit type.

6.1.3 Rock Wars

Rock Wars is a further extension of Complex RPS. We added more elements of real turn-based games: map topologies and unit building. The game draws its name from the “rocks” we added as passive objects, which simply serve as obstacles. Apart from an x and y coordinate pair they have no attributes or any actions. Their purpose is to hinder the movement of units, but they do not affect the attack actions. As the `go{North|East|South|West}` actions used in complex RPS already contained a condi-

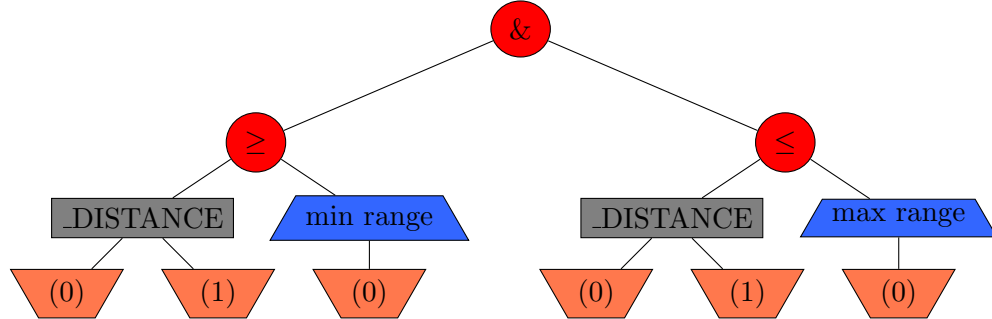


Figure 6.4: Adding range conditions to attack actions. The left subtree checks that the distance between attacker and target is greater than or equal to the minimum range of the attacker and the right subtree compares the same distance if it is lesser than or equal to the maximum attack range.

tion, that the target tile must be empty, no further modification was necessary in the model except a new ObjectClass *factory* had to be added. An instance for each player is placed on the opposing ends of a map. It has no attributes besides x and y but three create actions, one for each unit class. The create action for a factory consists of three parts: the cost condition, the condition of a free spawn tile, and finally the consequence for the actual unit creation. Figure 6.6 shows the complete action for a factory to spawn a new unit. With no starting units, the objective of the game now includes building the best unit combinations with a given budget of money to defeat the enemy units.

With *Rock Wars* we introduced the game- and player state (as described in section 5.16). The game state remains unused in this example, but the player state holds three variables for each player: a money indicator as a limiting factor for unit creation, and an x/y value, specifying the tile where the player factory would create a new unit. This was necessary as we used a procedural method to randomise the locations of the rocks on the map. On rare occasions this would place a rock at the factory spawn point, making it impossible for a player to create any units. Instead, we manually specified the location of the factory the new units' spawn point and excluded that location and the surrounding 3×3 area from the rock placement to ensure a clear starting location. Figure 6.5 gives an overall overview of the game.

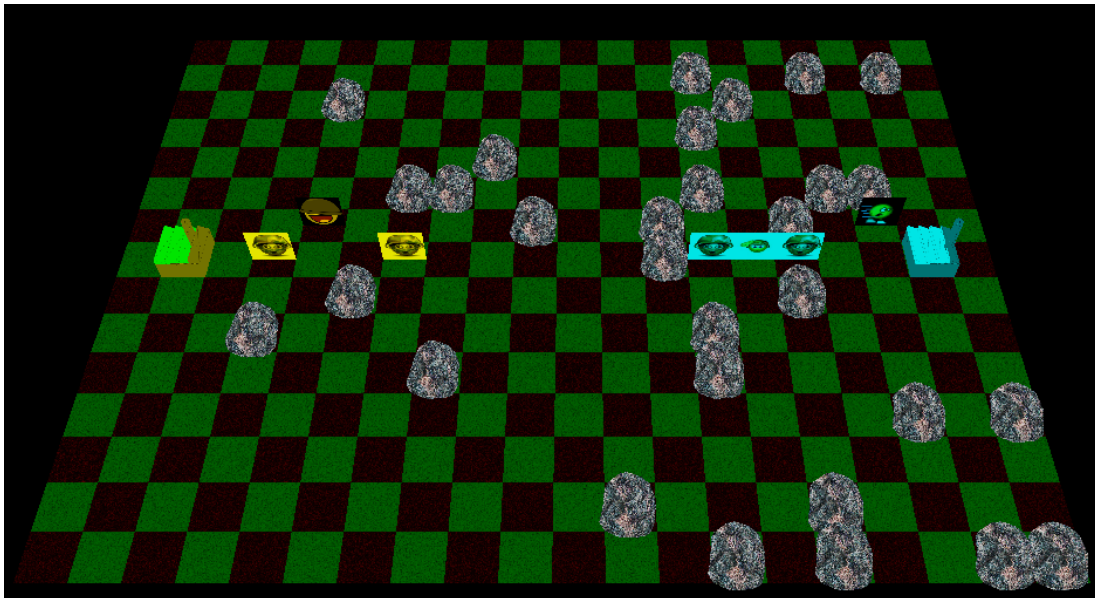


Figure 6.5: Screenshot of “Rock Wars”. The left part of the picture shows the factory of player one and a few created units; the right side of the screen shows respective area and units for player two. The screenshot also illustrates the rocks cluttered across the map.

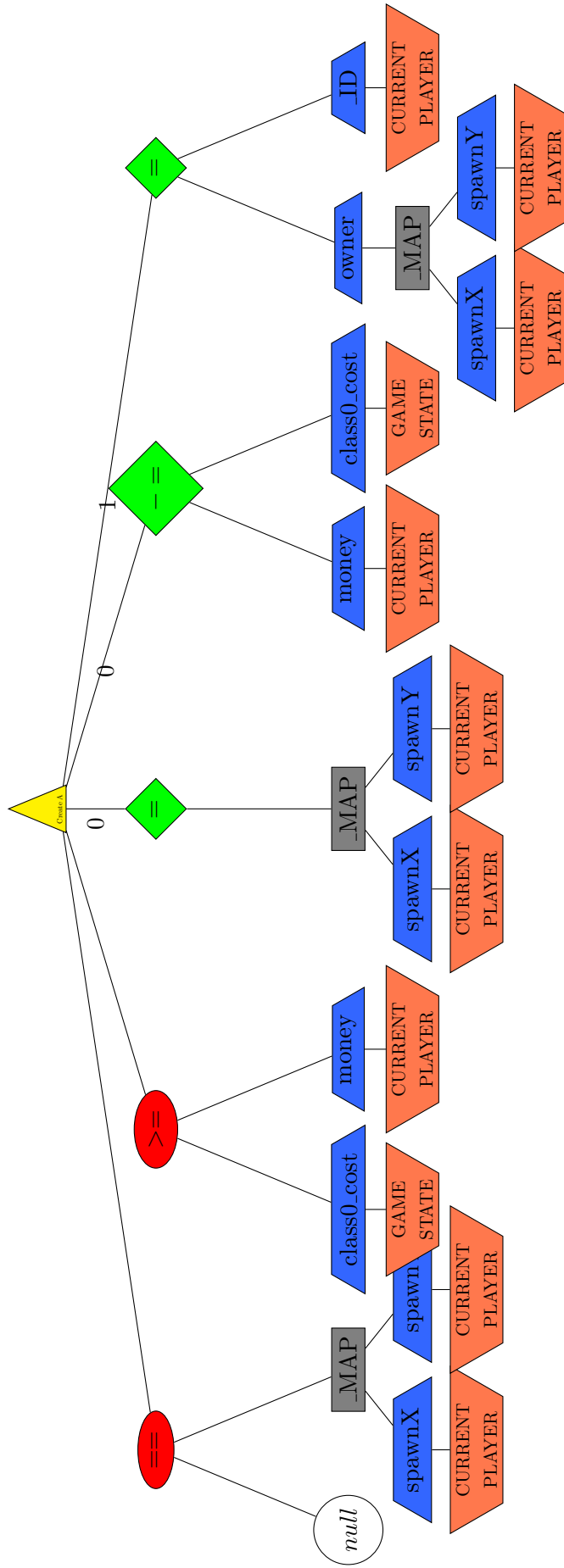


Figure 6.6: The detailed subtree for a "Create" action in "Rock Wars".

6.1.4 Dune 2

Dune 2 was our first approach to model an already published and commercially sold game. Our focus was to test if it was possible to model the complete game mechanics, ignoring the changes to playing experience and balance this might impose. Dune 2 was originally a real-time strategy game published by Westwood in 1993. It is often considered the first modern real-time strategy computer game.

The original game is loosely based on Frank Herbert’s novel *Dune* (182) where different parties fight over the control of a desert planet called *Arrakis* which is the only planet in the universe where the substance *Spice* can be harvested. While the novel spins a narrative around the substance Spice, the game reduces Spice to resource simply used to run a player’s economy: special units (called “harvesters”) roam the map, and collect Spice that has been placed on the map before game start. Upon the harvester’s return the collected spice is directly converted to money which in return the player can use to construct new buildings in his base or manufacture new units to wage war against his enemies on the map. We recreated this game mechanic in a simplified version of the game (as seen in figure 6.7). To place the Spice in an interesting way on the map, we used our map generator from our paper (183) published in 2012 using cellular automata. Section 6.2 will briefly present this approach.

6.2 Interlude: “Spicing up map generation”

Although the actual Strategy Description Language does not cover game elements besides game mechanics, the SGDL framework also features aspects that are needed to actually play the expressed games. For example, the integrated agents will be presented in chapter 7. In conjunction with our version of *Dune II* we would like to add the implementation of a map generator, that was published in 2012, to generate interesting maps (183). The implementation was specific to that game, but could be easily adapted to others. This experiment was conducted aside from our main research of generating strategy game mechanics, and is therefore presented in its own section.

Togelius et al. already explored the possibilities of a search-based map generator for the real-time strategy game *StarCraft* (120). They recognised that devising a single good evaluation function for something as complex as a strategy game map is anything but easy, so the authors defined a handful of functions, mostly based on distance and path calculations, and used multi-objective evolutionary algorithms to study the interplay and partial conflict between these evaluation dimensions. While providing insight into the complex design choices for such maps, it resulted in a computationally



Figure 6.7: Dune II in the SGDL engine.

6. SGDL IN PRACTICE

expensive map generation process. Furthermore, problems with finding maps that are “good enough” exist in all relevant dimensions. The map representation is a combination of direct (positions of bases and resources) and indirect (a turtle-graphics-like representation for rock formations), with mixed results in terms of evolvability.

Our approach is less complex, and based on an idea used by Johnson et al. for generating smooth two-dimensional cave layouts using cellular automata (CA) as presented in section 4.1.4. Their idea supports the particular design needs of a two-dimensional endless dungeon crawler game (129). Our map generator consists of two parts: the genotype-to-phenotype mapping and the search-based framework that evolves the maps. The genotypes are vectors of real numbers, which serve as inputs for a process that converts them to phenotypes, i.e. complete maps, before they are evaluated. The genotype-to-phenotype mapping can also be seen as, and used as, a (constructive) map generator in its own right. (The relationship between content generators at different levels, where one content generator can be used as a component of another, is discussed further in (184). The genotype-to-phenotype mapping is a constructive algorithm that takes an input as described in the following and produces an output matrix o . Based on tile types of the original Dune 2, the elements of o can assume the value 0 = SAND, 1 = ROCK, and 2 = SPICE. The matrix o is then later interpreted by the SGDL game engine into an actual game map.

The input vector is structured as followed (*mapSize* refers to the map’s edge length):

- n the size of the Moore-neighbourhood $[1, \frac{mapSize}{2}]$
- n_t the Moore-neighbourhood threshold $[2, mapSize]$
- i the number of iterations for the CA $[1, 5]$
- $w_{00}..w_{99}$ members the weight matrix w for the initial noise map $[0, 1]$
- s the number of spice blooms to be placed on the map $[1, 10]$

The generator starts with creating the initial map based on the values w . The 10×10 matrix is scaled to the actual map size and used as an overlay to determine the probability of a map tile starting as rock or sand. For each iteration i_n a CA is invoked for each map tile to determine its new type. If the number of rock tiles in the n -Moore-Neighbourhood is greater or equal than n_t the tile is set to *ROCK* in the next iteration.

The next step is the determination of the start zones, where the players’ first building will be placed. We always use the largest rock area available as the starting zones. The selection is done by invoking a 2D variant of Kadane’s algorithm (185) on o to find the largest sub-matrix containing ones. To prevent players from spawning too close to each other, we invoke Kadane’s algorithm on a sub-matrix of o that only represents the i top rows of o for one player, and only the i bottom rows for the other player. We let

6.2 Interlude: “Spicing up map generation”

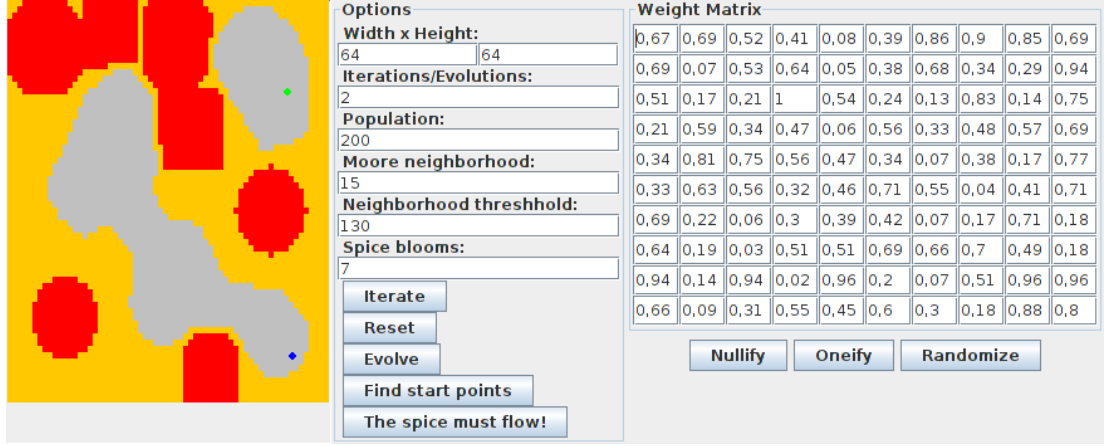


Figure 6.8: Screenshot of the generator application. The right pane lets the user input a seed matrix directly, or observe the result of the evolution. The middle pane can be used to either invoke the generator directly (“Iterate”) or start the non-interactive evolution (“Evolve”). The other buttons allow the user to go through the map generation step-by-step. The left pane shows a preview of the last map generated: yellow = sand, gray = rock, red = spice. The blue and green dot symbolise the start positions.

i run from 8 to 2 until suitable positions for both players are found. This operation ensures that one player starts in the upper half of the map and one in the lower. It also restricts us to maps that are played vertically, but this could be changed very easily. At this step we don’t assert that the start positions are valid in terms of gameplay. Broken maps are eliminated through the fitness functions and the selection mechanism.

The last step is the placement of the spice blooms and filling their surrounding areas. Since Kadane’s algorithm finds sub-matrices of ones, we simply clone o and negate its elements with $o_{nm} = 1 - o_{nm}$; whereas o_{nm} is the m -th member of the n -th row of o . We use the computed coordinates to fill the corresponding elements in o with spice. In order to make the fields look a bit more organic, we use a simple quadratic falloff function: a tile is marked as spice if its distance d from the center of the spice field (the bloom) fulfils the condition $\frac{1}{d^2} \geq t$. Where t is the width of the spice field multiplied by 0.001. We created a simple frontend application to test the generator. A screenshot with a basic description can be seen in Figure 6.8.

The genetic algorithm optimises a genome in the shape of a vector of real-numbers, using a fitness function we will describe in the following. Since a desert is very flat, there exists almost no impassable terrain, hence choke points (as introduced in (120)) is not a useful fitness measure. The challenge of choke points was instead replaced by

6. SGDL IN PRACTICE

the assumption that passing sand terrain can be rather dangerous due to sandworms. Furthermore, it should be ensured that both players have an equally sized starting (rock) zone and the distance to the nearest spice bloom should be equal. All values were normalised to $[0, 1]$. To summarise, the following features were part of the fitness function:

- the overall percentage of sand in the map s
- the Euclidean distance between the two starting points d_{AB}
- the difference of the starting zones' sizes Δ_{AB} (to minimise)
- the difference of the distance from each starting position to the nearest spice bloom Δd_s (to minimise)

Apart from these criteria a map was rejected with a fitness of 0 if one of the following conditions was met:

- There was a direct path (using A^*) between both starting positions, only traversing rock tiles. (Condition c_1)
- One or both start positions' size was smaller than a neighbourhood of eight; and no reasonable space for building a base would be available. (Condition c_2)

The resulting fitness function was:

$$f_{map} = \begin{cases} 0 & \text{if } c_1 \vee c_2, \\ \frac{s+d_{AB}+(1-\Delta_{AB})+(1-\Delta d_s)}{3} & \text{else} \end{cases} \quad (6.1)$$

In other words: the average of the components if the map passed the criteria, 0 otherwise.

With appropriate parameters, we were able to generate several maps that resembled the style of the original Dune 2 maps. From an aesthetic point of view, the maps look interesting enough to not bore the player and remind them of the original Dune 2 maps, while still presenting fresh challenges. Three example maps can be seen in figure 6.9

6.3 Summary

This chapter presented a few complexer games expressed in SGDL. We also introduced the game *Rock Wars* which will form the base for games used in the experiments presented in the following chapters. Rock Wars is simple turn-based strategy game which features some basic elements of all strategy games: unit production, tactical combat, a map with obstacles, and a simple economic model. We also presented an outlook on future games and the extended use of map properties with the discussion of

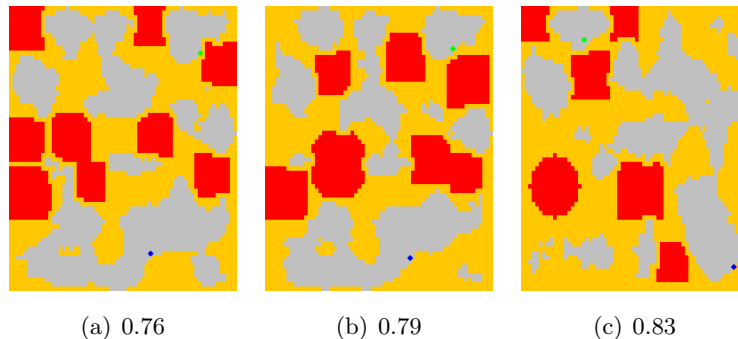


Figure 6.9: Three examples of generated Dune II maps, captioned with their fitness values. All three maps were created through the evolutionary process. The yellow areas denote sand, the grey areas are rocky terrain where the players can build on, and the red areas represent “Spice”, a resource that players may harvest and use as currency.

a map generator for a version of the popular real-time strategy game *Dune 2* modelled in SGDL.

In the following, we will now shift the focus on the automatic generation of game mechanics using several approximations of “interesting” gameplay for an evolutionary algorithm. All generated games are expressed in SGDL. But before introducing the experiment setup, another component is introduced that enables us to make use of simulated gameplay: the SGDL agent framework.

6. SGDL IN PRACTICE

Chapter 7

Automated Gameplay

This chapter presents a different part of the SGDL framework, the component of artificial general gameplayers. We present a series of agents based on different techniques from the field of *machine learning*, and set them into context with the example games 6.1 and their relative performance. Ultimately, the agents were tested as opponents for human players in an online experiment. The resulting data is presented in section 7.4. This chapter presents work of a secluded project done and and previously published by Nielsen and Jensen as part of the overall SGDL project (47, 186).

The utilisation of simulated gameplay for fitness evaluations of strategy games requires agents which are capable of playing said games at an adequate skill level like a human player would. Optimally, agents would be able to imitate playing styles of different playing preferences. In this section, we address the problem of *general* strategy game playing. This implies agents which can proficiently play a wide variety of strategy games and scenarios, not just a single game or scenario. (The definition allows the agents some time to adapt to the particular game and scenario.) The aim is to create agents which can play as many different strategy games as possible, rather than a collection of useful domain specific “hacks”. At the same time, the range of games our agents are supposed to handle is considerably more constrained than the range of games expressed by the Stanford GDL, used in the General Game Playing Competition (172); all of the games (also referred to as *models* hereafter) considered in this section are variations of *Rock Wars*, as presented in section 6.1.

Six different agent architectures (plus variations) are implemented. These are based on techniques that have been successfully applied on various forms of game-related AI problems: playing board games (MinMax, Monte Carlo Tree Search), autonomous agent control (Neuroevolution, potential fields), commercial video game AI (finite-state

7. AUTOMATED GAMEPLAY

machines) and strategy selection (classifier systems). Two different random agents are also implemented for comparison. For those architectures that are based on some form of learning algorithm, relatively extensive training is performed for each agent on each model. We ensured, that each architecture was provided with an equal amount of training time. It should be added, that a heuristic described in section 9.2 was excluded from the experiments, because all agents were able to defeat it constantly. Said heuristic was also created before our agent tests were concluded, hence its main purpose was to be a surrogate until the agent development was finalised. Then, the random agents provided a sufficient baseline.

Two different kinds of evaluation of the agents were undertaken. The first was an extensive payout of every agent against every non-learning agent on all of the defined models. From this, the relative performance (in terms of winning/losing games) of trained agents against each other can be gauged. The second evaluation form was interactive: human players played against at least two different agents each, using different game models, and answered a short survey about their preferences between those agents. From this we can gather both objective data (which agent was best against human players?) and subjective (which was best-liked?) about these agents' interactions with humans.

The questions we are addressing in this section, and which we claim to be able to answer at least partially, are the following:

- Is it possible to construct agents that can proficiently play not just one but a range of different strategy games?
- How can we adapt some specific AI techniques that have been successful on other game-related problems to work well with strategy games?
- Which of these techniques work best in terms of raw performance?
- Which of these techniques make for the most entertaining computer-controlled opponents?

7.1 The game, the agents and the assigner

The overall principles taken from *Rock Wars* (as explained in section 6.1.3) remain the same. Each player starts with a non-moveable building which possesses the ability to spawn new units. All units take 1 turn to produce. Each unit costs a certain amount of a resource based on the SGDL model. The players' resources and the unit costs are tracked within the SGDL *game state*. The template for this game state is read from the loaded SGDL model, i.e. the unit costs and starting resources depend on the loaded

model. Each unit has one action per turn, and may select any possible action defined for its class. One of the possible actions could be to move to an unoccupied map tile or shoot at an enemy unit. If a unit loses all its health points it is removed from the game. If a player loses all his units (besides his factory) and has no resources left to produce new units, his opponent wins the game. Should the game take longer than 100 turns, the game is a draw.

The agents in the study were based on a hierarchical agent framework named the “Commander framework” based on the *Intelligent Agent Architecture* by Russell and Norvig (1987), as can be seen in figure 7.1. The framework consists of a commander entity on the highest logical layer, and sets of unit and building entities. Objects in a game belonging to an agent are linked to these sets of unit and building entities in the framework, while the commander entity is a strategic entity only. Thus the framework is separated in a higher level layer, called the strategic layer, and a lower level layer, called the unit layer.

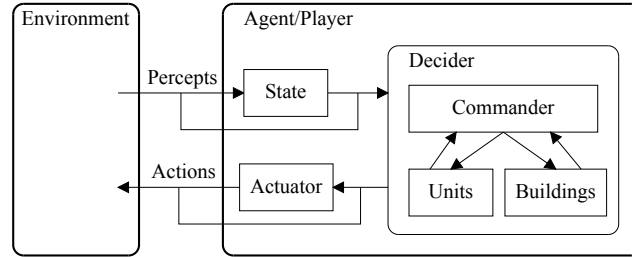


Figure 7.1: The Commander framework used for the agents in the study.

The framework was designed such that it can be used by game tree based techniques as well as multi-agent techniques. This is possible by the two-way communication between all types of entities, which allows either a commander to control units and/or buildings, units and buildings to act autonomously, or any other combination necessary.

The communication with the SGDL game engine is maintained through two utility systems named State and Actuator. Because the SGDL model of a game consists of an unknown quantity of percepts and actions with unknown properties, it is beneficial for non game tree based techniques to use a system that categorises percepts and actions into sets of known quantities and properties. An agent may test if a specific action is possible at any time during his turn by testing its conditions. The SGDL framework also supports supplying all possible actions for a certain unit. This is done through testing all the conditions of all of the actions that object could theoretically

7. AUTOMATED GAMEPLAY

invoke. Actions which require an additional object (target object) are tested with all objects that are also on the map. Because of a pre-set constraint, no action in our games requires more than two objects (acting object and target object) we can limit the search to one extra object. Otherwise conditions would have to be checked against all permutations of objects on the battlefield. Although an agent with more domain knowledge might apply a faster and more efficient way to select actions, the agents described in this chapter rely on the set of possible actions created through a “brute-force” search method. The framework simply tests all `WorldObject` and actions combinations, and reports the possible actions back to the agent. If an agent submits an action to the framework that is not possible, it would simply get denied.

The State system consists of a set of general information that captures a subset of the percepts thought to be the minimum amount of information necessary for agents to react meaningfully to the game environment. Included in the set of states are the type of class of the unit, its health, the distance, angle and relative power of the nearest three opponents, the distance and angle of the nearest two obstacles and the distance and angle of the opponents building. The Actuator system uses a one ply game tree search in order to determine the effect of all given actions, and categorises them into a finite set of actions with known effects. Included in the set of actions are the attack actions that do the most damage to opponents, actions that kill opponents and actions that cause movement in one of eight possible directions. The disadvantage of these systems is that information is no longer complete given the categorisations made, but they decrease the search space significantly; a requirement for many techniques to do meaningful searches.

Another utility function was developed for commander entities, which can provide additional information in terms of a relative measurement of the power of unit objects relative to each other and can assign orders to unit entities on the lower layer. Relative power information is gained through short simulations of the unit object types against each other, where power is based on the steps it takes for one to kill another. The order assignment is done through a neuroevolutionary approach based on NEAT (165). A bipartite graph that consists of the set of units belonging to an agent are fully connected to a set of units that belong to the enemy. Each edge of the bipartite graph is weighted by a neural network evolved through NEAT with a set of information relevant to each edge, i.e. distance, health and relative power measurement between the units connected. Assignments are determined by hill climbing, where the highest valued edges of the bipartite graph are selected for each unit that requires an assignment.

To test the agents’ flexibility with different SGDL models, five distinct models were

created to represent different strategy gameplay aspects. As seen in table 7.1, the models were named *chess*, *shooter*, *melee*, *rock-paper-scissor (RPS)* and *random*. All models are variations of *Rock Wars* (as described in section 6.1).

- *Rock-paper-scissors (RPS)*: a balanced strategy game where each unit can do heavy damage to one other class of unit, light damage to another and no damage to the third. This mirrors a popular configuration in strategy games where tanks are effective against foot soldiers, foot soldiers against helicopters and helicopters against tanks. All units have a movement of one.
- *Melee*: Similar to the RPS model, but all units have an attack range of one, forcing them to chase and entrap each other.
- *Shooter*: Perhaps to the model that is most similar to a standard strategy game. Shooter has 3 different classes, a sniper, a soldier and a special operations agent (special ops). The sniper has high range, medium damage and low health, the soldier has medium range, low damage and high health and the special ops has low range, high damage and medium health.
- *Random*: Units are only able to move one step, and the cost, health, ammo and damage of each class against all others is randomised for every game.
- *Chess*: A simplified chess game with unit movements and capabilities inspired by the rook, knight and bishop pieces.

The properties of all the models used are summarised in table 7.1. There was a limit of 100 turns per game and a limited amount of units which could be built based on their cost. The players started with an equal random amount of money that could be spent on units. The games are turn based, and there is no fog of war. The models described above determine the rules of each game. Any action, be it movement, shooting or similar, constitutes a turn, and so does doing nothing. Units are symmetric for the players in all games regardless of models and maps.

7.2 Agents

We created eleven agents based on several different techniques as presented in table 7.2. The non-learning agents' main purpose was to serve as training partners for the evolving agents, but they were also included in our experiments. The following sections will cover the details of each agent implemented.

7. AUTOMATED GAMEPLAY

Table 7.1: Unit properties for SGDL models

| Unit properties | SGDLs | | | | |
|-----------------|---------|---------|--------|-------------|--------|
| | Chess | Shooter | Melee | RPS | Random |
| Random cost | ✓ | × | ✓ | ✓ | ✓ |
| Random health | ✓ | × | ✓ | ✓ | ✓ |
| Random ammo | ✓ | × | ✓ | ✓ | ✓ |
| Random damage | ✓ | × | ✓ | ✓ (special) | ✓ |
| Random range | × | × | × | ✓ | ✓ |
| Movement note | Special | 1-step | 1-step | 1-step | 1-step |

Table 7.2: Agents in the study

| Agent name | Purpose |
|------------|--|
| Random | Opponent in agent test |
| SemiRandom | Training, Opponent in agent test |
| FSM | Training, Opponent in agent test, Human play testing |
| NEAT | Agent versus Agent testing, Human play testing |
| NEATA | Agent versus Agent testing |
| MinMax | Agent versus Agent testing, Human play testing |
| MCTS | Agent versus Agent testing, Human play testing |
| PF | Agent versus Agent testing, Human play testing |
| PFN | Training |
| XCS | Agent versus Agent testing |
| XCSA | Agent versus Agent testing |

7.2.1 Random action selection

Two agents relying on random action selection were created in this study to train the evolving agents and to provide a performance baseline. Both agents are capable of fitting into a multi-agent framework, as the logic executes on the level of the individual unit and not at a strategic level. These two agents are the following;

- Random agent
- SemiRandom agent

The Random agent selects a random action from the set of possible actions given by the Actuator, resulting in random behaviour. The SemiRandom agent is designed to move randomly but use the best available offensive action, thus making it an offensive

but largely immobile opponent. The agent uses the Actuator as well, which guarantees that the most effective actions in the game are used. As can be seen in figure 7.2, a *finite-state automaton* or *finite-state machine* (188) is used to determine the action to perform.

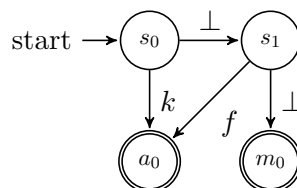


Figure 7.2: Finite-state automata of the SemiRandom agent units

7.2.2 Finite-state machine

Similar to the random agents, the *finite-state machine* (FSM, as described in section 4.3.2) agent was created to provide a performance baseline and a training partner for other agents. It utilises a finite state machine architecture with movement selection based on a local breadth first search. Figure 7.3 shows the structure of the automaton for the units, where the Actuator is used for action selection analogous to the random agents. The FSM agent is an better opponent than the Random agent, but it requires hand-coded domain knowledge about the game model.

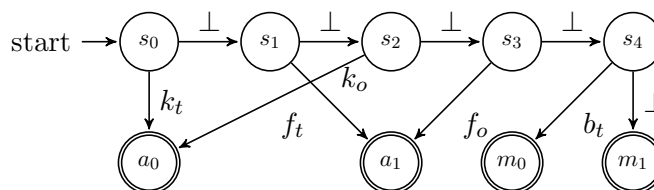


Figure 7.3: Finite-state automaton of the FSM agent's units

Congruent to the SemiRandom agent, an ordered sequence $\langle s_0, \dots, s_4 \rangle$ of transitional states is traversed. Unlike the SemiRandom agent, the FSM agent requires a hostile target for each individual sub-agent in order for a search to be successful; it is also used for attack selection as the target is prioritised. The accepting states are $\{a_0, a_1, m_0, m_1\}$, and are given by the Actuator. Since the FSM agent favours a targeted unit, the kill and attack conditions k and f are subdivided into k_t , k_o and f_t , f_o where t is the target, and o is any other unit. The breadth-first search is designated as

7. AUTOMATED GAMEPLAY

b_t , where the b_t condition is true if the breadth-first search is able to find a path towards the target. In this case the accepting state m_0 selects the first movement action which leads along this path.

The breadth first search is local because it has a limited amount of movement actions that can be searched; an *expansion limit*. In order for the search to be effective, a distance heuristic was applied on the ordered sequence used for expansion of moves in the search. When the limit has been reached the search terminates and the action with the shortest distance to the target is executed.

7.2.3 Neuroevolution of augmenting topologies

The *Neuroevolution of Augmenting Topologies* (NEAT, as described in section 4.3.4) agents are based on the evolutionary algorithm for neural networks developed by Stanley and Miikkulainen (165). This algorithm has previously been used with success for evolving agent controlling neural networks in, but not limited to, shooter games and racing games. The technique is a form of *topology and weight evolving artificial neural network*, such that it not only optimises weights in a neural network, but also constructs the network structure automatically via artificial evolution. Within the NEAT agents, action selection is based on an artificial neural network that has been trained through machine learning using evolution. A fitness function evaluates the performance of genomes in a population, and the fittest members are subsequently selected for the creation of new members by combining their genetic information through a cross-over genetic operator (189, 190).

Given the nature of artificial neural networks which can approximate any function given an arbitrary large network (191), and a topology which evolves to a functional structure, the agents are able to learn general gameplaying depending only on the fitness function. However, in this implementation the State and Actuator utilities were used to simplify and normalise the number of inputs and outputs. This means that the technique operates on a subset of the actual state and action space of the games, as was discussed in section 7.1.

The following two agents have been created that use an artificial neural network evolved through NEAT:

1. NEAT agent (Neuroevolution of Augmenting Topologies agent)
2. NEATA agent (Neuroevolution of Augmenting Topologies agent with Assigner)

The fitness function used for the successful NEAT agent - out of several investigated in the study - can be seen in equation (7.1). Here w is the amount of wins, l is the

amount of losses, and d is the amount of draws. Each genome is tested in six games against three opponents, and evaluated using this function. The function was made to force the agent into avoiding draw games and prioritise winning. However, its behaviour is consistently hesitant to pursuing opponents, and it instead waits for the opponent to approach.

$$f_{NEAT}(a_i) = \frac{w}{w + l + d} \quad (7.1)$$

Several fitness functions were investigated, using more information than just the winrate as above, such as including a normalised distance measure to encourage a behaviour which engages opponents more; a flaw of the above fitness measure. However, the winrate decreased when using these information additions, even though the behaviour of the agent became more as desired in that it would aggressively pursue the opponent. The problem might be caused by conflicting objectives; pursuing seems to counteract its ability to win, i.e. the agent sends units on suicide missions. Equation (7.1) received the largest amount of victories, and was thus chosen for training.

The NEATA agent has one variant which can be seen in equation (7.2). Here s is the number of successful orders carried out by units given out by the Assigner (see section 7.1), b is the number of kills that were not given by orders and q the number of failures to perform orders, e.g. the death of a friendly unit. It is normalised to the number of units which have received orders denoted by u .

$$f_{NEATA}(a_i) = \left| \frac{s + \frac{b}{4} - \frac{q}{4}}{u} \right| \quad (7.2)$$

The function drives the agent to evolve a behaviour that can successfully kill the hostile unit which has been designated as a target, and potentially kill any other hostile unit it encounters on its way. Because of the negative value given for a failure, it also attempts to avoid destruction while carrying out the orders.

7.2.4 MinMax

The MinMax agent is based on the classic MinMax algorithm with alpha-beta pruning (187), which is one of the simplest and most popular algorithms for playing games and which has achieved considerable success on board games with low branching factors, like Chess and Checkers. It has been previously described in section 4.3.1.1.

When implementing game tree search-based agents, it was decided that every branch in the search tree represents a set of actions, one action for each friendly moveable unit. We will refer to such a set as a *MultiAction* in the following.

7. AUTOMATED GAMEPLAY

Both game tree agents run in the Commander-Structure within the Agent framework as seen in figure 7.1. After a search has been executed, the best MultiAction is determined and its actions are distributed to the units. Neither the MinMax nor the Monte Carlo Tree Search (MCTS) agent (presented in the following subsection) use State or Actuator as the algorithms search ahead through potential actions.

Implementing MinMax with alpha-beta pruning into an agent required the modification of the algorithm as its runtime complexity grows rapidly with the branching factor and depth of the tree. Since the amount of units and actions are unknown in the model, a limit had to be placed on the amount of MultiActions possible. Without a limit, too many MultiActions could cause the agent to play poorly or become dysfunctional.

To limit the MultiActions, depth first search (DFS) is performed on a tree with moveable units and actions. Refer to figure 7.4 as an example, where U_x is moveable unit x , and U_{xy} is its action y . The DFS is limited to only choose from the child nodes of its current best choice, starting from the root. For example, should the DFS choose $U_{1attack}$ over U_{1flee} , it would then have to choose between $U_{2attack}$ and U_{2flee} . When a leaf is found, the MultiAction (built from path of actions selected from root to leaf) is saved, and the DFS moves one step back towards the root and selects again.

To help guide the selection of MultiActions, a heuristic is needed to evaluate how advantageous a game state for an agent is. The heuristic used is the same board evaluator used within MinMax when a maximum depth is reached. Constructing a heuristic for changing models proved to be a challenge as units in class 1 might have different actions and attributes in different SGDL models. A static heuristic would be infeasible, as the parameters and actions highly affect a unit's effectiveness, and instead a neural network was evolved using NEAT with the inputs: the health ratio between friendly and hostile units and the average euclidean distance to nearest enemy, weakest enemy, friend and enemy building. The single output of the network is how favourable the board configuration is. The neural network was evolved by evaluating its performance based on its win rate against a set of test agents of varying behaviour.

To help MinMax predict the actions of the enemy, even without units currently on the board, the buildings and their actions had to be implemented in the tree seen in figure 7.4. This would further increase the amount of MultiActions if every action of the building is represented. It was therefore chosen to limit the buildings' actions to one. The unit produced in the search tree would be decided randomly (see figure 7.5).

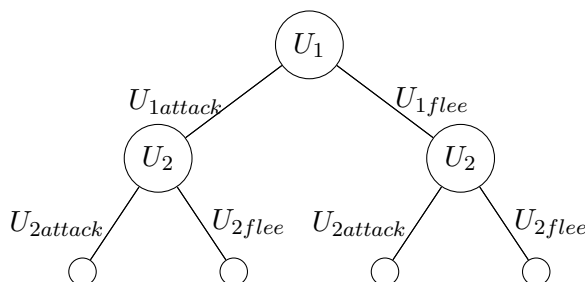


Figure 7.4: Action tree used to find MultiActions, performed at each node in the search tree.

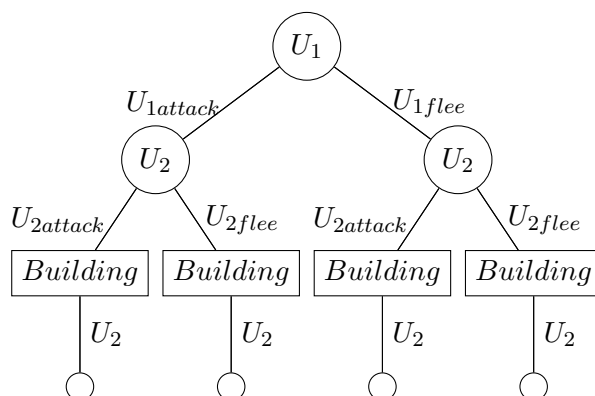


Figure 7.5: Action tree illustrating the use of buildings. The building nodes are allowed only one child each to limit the complexity.

7.2.5 Monte Carlo Tree Search

The Monte Carlo Tree Search (MCTS, as described in section 4.3.1.2) agent is based on the MCTS algorithm, which has recently seen considerable success in playing Go (144). Even though MCTS is known for handling trees with large branching factors, the branching factor of most SGDL models is drastically higher than Go. Considering this issue, the MCTS agent was implemented with the same MultiAction filter as the MinMax agent. Once MCTS was implemented, it could be observed that the algorithm only produced 100-150 Monte-Carlo simulations per second due to the computational overhead of cloning in Java. Part of this problem is caused by our naïve implementation where the whole gamestate (including map) is cloned for every node. As a solution to this, MinMax’s board evaluation function was used instead of the *play-out* phase. Retrospectively, a (potentially better) solution would have been an adaptive cloning

7. AUTOMATED GAMEPLAY

mechanics that only records the object that have changed based on an action. However, the regular play-out outcome z_i of simulation i is replaced with a state value approximation, which is backpropagated towards the root as normal.

Different exploration policies, as described in section 4.3.1.2 have been tested with our agent: MC-RAVE produced the highest win rate with k -value of 10. A pattern emerged during the experiments. UCT-RAVE ($c = 0.1$) scored a worse win rate than MC-RAVE against FSM (38.95% vs. 45.96%), SemiRandom (32.63% vs. 37.89%) and Random (56.49% vs. 66.14%) with p-values 0.05, 0.16, and 0.02. For the MCTS agent, UCT ($c = 0.1$) performed worse than UCT ($c = 0$). When MCTS iterations were forced to explore, rather than focusing on the early best looking child nodes, the win rate was decreasing. This is most likely caused by either a too low iteration count and/or the use of a board evaluation function, replacing the regular play-out phase. If the reason is a too low iteration count, giving the algorithm more time to think (more than one second) would increase the iterations and as a result reward the act of exploring child nodes of less immediate interest. On the other hand, raising the time constraint to more than a second is undesirable, as it affects the experience of a human player in a negative way; even though we are only considering turn-based games. Replacing the playout-phase with a neural network evolved using NEAT, might affect the Monte Carlo-value by setting it close to its exact value even after only a few iterations - and exploration would therefore become obsolete.

7.2.6 Potential fields

The potential field (PF) agent developed in this chapter is similar to the multi-agent potential field approach which has recently shown good performance in some real-time strategy games (11, 192, 193). The potential of a point on the map of the game is expressed as in equation 7.3, where P is a set of potentials and $w(p_n)$ is a function that maps a weight to potentials and pheromone trails; inspired by the behaviour of colonies of insects. Potential functions take a distance from the x and y variables and the position of the potential p_i using the euclidean distance. A pheromone trail is given as k , which is a trail of pheromone left by each unit, where each individual pheromone is weighted inside a pheromone function, such that they decrease in strength over time. As such, they serve as a negative trail of potentials with decreasing effect, and forces the units to move in paths not previously taken. There is a potential for each object on the map which contains the position of the object, and additional input variables not given in equation 7.3 that apply specifically for the various potentials depending on their purpose.

$$f(x, y) = \sum_{i=1}^{|P|} (p_i(d) w(p_i)) + (k(x, y) w(k)) \quad (7.3)$$

By using this formula to calculate the potential of a point, it is not necessary to calculate the global field of the map. Each unit contains its own potential field, which is calculated for the legal moves that it can use to make in its turn, and in order to keep the pheromone trail local to the unit.

$$p_{hostileunit}(d) = \begin{cases} \left(\frac{|m-d|}{m}\right)^2 power, & \text{if } power > 0; \\ -\left(\frac{|m-d|}{m}\right)^2 \frac{1}{2}, & \text{otherwise.} \end{cases} \quad (7.4)$$

As there is a potential for each object, and given that there are different types of objects, multiple potential functions such as the one in equation (7.4) were formulated. The above function creates a potential for hostile units, where m is the maximum distance on the map, d is the distance between the sub-agent and the hostile unit, and $power$ is the relative power measure given by the Assigner utility. Various other functions are given for friendly units, buildings obstacles etc.

An additional agent named PFN with a negative sign in equation (7.4) was used for training, as it would avoid enemy contact and require the agents trained to learn how to give chase.

7.2.7 Classifier systems

Another two agents were implemented based on eXtended Classifier Systems (XCS) (194): a regular *XCS agent* and a XCSA (eXtended Classifier System using Assigner) agent using the Assigner for orders. Both agents operated within the units and did not use the Commander entity in the Commander Architecture as seen in figure 7.1. All units for both agents shared the same XCS structure, resulting in shared information about the environment.

The XCS classifier system builds on Holland's Learning Classifier Systems (190) (LCS) which is a machine learning technique that combines reinforcement learning and evolutionary computing. A classifier system creates rules through evolutionary computing and tries to predict the external reward by applying reinforcement learning through trial and error. LCS changes the fitness of the rules based on external reward received, while XCS uses the accuracy of a rule's prediction.

To adapt to changing SGDL models, the XCS structure was slightly modified. In Wilson's XCS a covering occurs when the number of classifiers in the *Match set* is below

7. AUTOMATED GAMEPLAY

a threshold. Following Wilson’s advice by populating through covering, setting such a threshold can be difficult with changing SGDL models, as the number of actions is unknown. A low threshold resulted in the Match Set filling up with move actions, as attack actions were met later in the game when opponents were engaged. The threshold was changed to force the XCS structure to have at least one classifier for each possible action in the current environment. In some SGDL models, unit attributes changed over different games, therefore classifiers representing illegal moves are removed from the Match Set.

To reward the *Action Sets*, the XCS agent had to wait for the game to end, in order to receive a *won*, *draw* or *loss* feedback from the game. All Action Sets a were then rewarded through equation 7.5. There, Ω is a function returning 1 for win and 0 for loss or draw, D is the average Euclidean distance to nearest enemy and D_{max} is the maximum possible distance.

$$r(a) = 1000\Omega + 500(1 - \frac{D}{D_{max}}) \quad (7.5)$$

The XCSA agent utilised the Assigner’s order updates throughout the game and rewarded (see equation 7.6, where Λ is the euclidean distance to the target) immediately once it was told if the action was good or bad. Using equation 7.6, each order event signal was rewarded differently. Upon receiving a successful event signal, the $reward_{order}$ was set to 1000. A mission cancelled or failed signal led to $reward_{order}$ being 0, and should the unit receive the event signal of killing an enemy outside the ordered target, 500 were set for $reward_{order}$.

$$r(a) = reward_{order} + 500(1 - \frac{\Lambda}{D_{max}}) \quad (7.6)$$

7.3 Results of agent versus agent training

Before evaluating them, most of the agents needed to be trained in order to perform well. Training was undertaken separately for each agent on each model, but always against all three opponents. It was ensured that all agents were trained for the same amount of time (for fairness), and long enough, so a performance convergence could be observed for each agent type.

The performance of the agents against the FSM, SemiRandom (SR) and Random (R) agent in terms of the ability to win games on the different SGDL models and maps was measured through experimentation. Nearly a million games in total were run, concluding in the results presented below.

7.3 Results of agent versus agent training

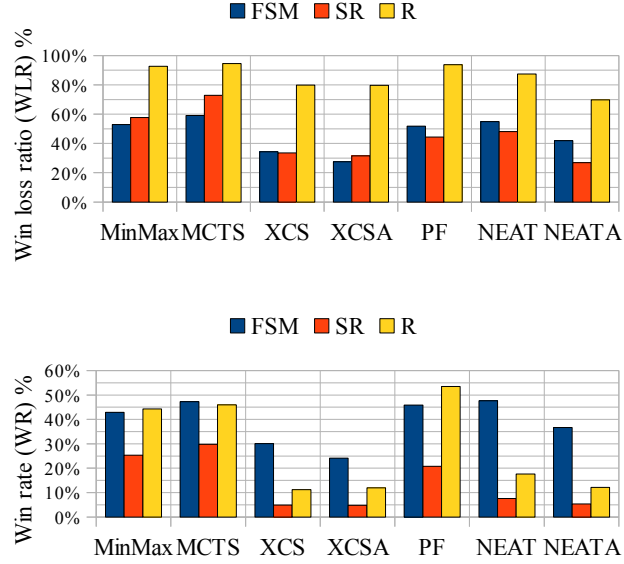


Figure 7.6: Summary of agent versus agent results

The results are analysed in terms of the *win rate* (WR) and the *win loss ratio* (WLR), where the win rate is given as $\frac{w}{w+l+d}$ and the win loss ratio is given as $\frac{w}{w+l}$. Here w is the amount of games won, l is the amount of games lost and d is the amount of games that ended in a draw. The win loss ratio ignores the draws, in order to focus on the ratio of win/loss against opponents, but must be seen with respect to the win rate which is the win ratio in terms of total games played.

In table 7.3 the following terms are used; *Opp.* refers to the opponent agents, W refers to won games, L refers to lost games, D refers to draw games, WLR refers to the win loss ratio, WR refers to the win rate. The standard deviations are given in table 7.4 where the following terms are used; $\overline{\sigma_{WLR}}$ refers to the mean standard deviation of the win loss ratio and $\overline{\sigma_{WR}}$ refers to the mean standard deviation of the win rate. The terms $\sigma_{\overline{\sigma_{WLR}}}$ and $\sigma_{\overline{\sigma_{WR}}}$ denote the standard deviation of the population of the standard deviations given in the above means. This is necessary as the samples are divided on different models and maps.

In total, 8550 games were played for the adversarial search based agents MCTS and MinMax and 55575 for the other agents. The previously mentioned computational complexity of the search based agents required us to make this differentiation in order to perform the experiments in a reasonable time frame. The variation between samples, as seen in table 7.3, is highest with MinMax and MCTS given the smaller sample size,

7. AUTOMATED GAMEPLAY

Table 7.3: Summary of agent versus agent results

| Agent | Opp. | W | L | D | WLR | WR |
|--------|------|-------|-------|-------|--------|--------|
| MinMax | FSM | 3667 | 3264 | 1619 | 52.91% | 42.89% |
| MinMax | SR | 2164 | 1584 | 4802 | 57.74% | 25.31% |
| MinMax | R | 3787 | 297 | 4466 | 92.73% | 44.29% |
| MCTS | FSM | 4038 | 2799 | 1713 | 59.06% | 47.23% |
| MCTS | SR | 2549 | 947 | 5054 | 72.91% | 29.81% |
| MCTS | R | 3930 | 225 | 4395 | 94.58% | 45.96% |
| XCS | FSM | 16691 | 31865 | 7019 | 34.37% | 30.03% |
| XCS | SR | 2695 | 5337 | 47543 | 33.55% | 4.85% |
| XCS | R | 6226 | 1570 | 47779 | 79.86% | 11.20% |
| XCSA | FSM | 13395 | 35280 | 6900 | 27.52% | 24.10% |
| XCSA | SR | 2653 | 5771 | 47151 | 31.49% | 4.77% |
| XCSA | R | 6622 | 1679 | 47274 | 79.77% | 11.92% |
| PF | FSM | 25505 | 23643 | 6427 | 51.89% | 45.89% |
| PF | SR | 11526 | 14461 | 29588 | 44.35% | 20.74% |
| PF | R | 29711 | 1976 | 23888 | 93.76% | 53.46% |
| NEAT | FSM | 26461 | 21741 | 7373 | 54.90% | 47.61% |
| NEAT | SR | 4172 | 4496 | 46907 | 48.13% | 7.51% |
| NEAT | R | 9759 | 1393 | 44423 | 87.51% | 17.56% |
| NEATA | FSM | 20391 | 28308 | 6876 | 41.87% | 36.69% |
| NEATA | SR | 2973 | 8122 | 44480 | 26.80% | 5.35% |
| NEATA | R | 6726 | 2901 | 45948 | 69.87% | 12.10% |

but it is low in general for all agents.

As can be seen in figure 7.6 and table 7.3, the MCTS, MinMax, PF and NEAT agents have a WLR near or above 50%. XCS, XCSA and NEATA have a WLR lower than 50% on all opponents other than the Random agent. Only the MCTS and MinMax agent were able to defeat the SemiRandom agent. To our surprise, the SemiRandom agent demonstrated quite good gameplay on most models despite its naive approach. It tends to gather its units in small clusters with effective selection of offensive actions based on the finite-state automaton logic.

With regards to WR, most agents had a performance less than 50% against all opponents because of draw games. The MinMax, MCTS and PF agents have the

7.3 Results of agent versus agent training

Table 7.4: Standard deviantions of agent versus agent results

| Agent | Opp. | $\overline{\sigma_{\text{WLR}}}$ | $\overline{\sigma_{\text{WR}}}$ | $\sigma_{\overline{\sigma_{\text{WLR}}}}$ | $\sigma_{\overline{\sigma_{\text{WR}}}}$ |
|--------|------|----------------------------------|---------------------------------|---|--|
| MinMax | FSM | 0.05 | 0.04 | 0.02 | 0.02 |
| MinMax | SR | 0.08 | 0.04 | 0.04 | 0.02 |
| MinMax | R | 0.04 | 0.04 | 0.03 | 0.02 |
| MCTS | FSM | 0.04 | 0.04 | 0.02 | 0.02 |
| MCTS | SR | 0.07 | 0.04 | 0.02 | 0.02 |
| MCTS | R | 0.04 | 0.04 | 0.04 | 0.02 |
| XCS | FSM | 0.02 | 0.02 | 0.01 | 0.01 |
| XCS | SR | 0.05 | 0.01 | 0.01 | 0.00 |
| XCS | R | 0.04 | 0.01 | 0.03 | 0.01 |
| XCSA | FSM | 0.02 | 0.01 | 0.01 | 0.01 |
| XCSA | SR | 0.04 | 0.01 | 0.02 | 0.01 |
| XCSA | R | 0.05 | 0.01 | 0.03 | 0.01 |
| PF | FSM | 0.02 | 0.02 | 0.01 | 0.01 |
| PF | SR | 0.03 | 0.01 | 0.01 | 0.01 |
| PF | R | 0.01 | 0.02 | 0.01 | 0.01 |
| NEAT | FSM | 0.02 | 0.02 | 0.01 | 0.01 |
| NEAT | SR | 0.04 | 0.01 | 0.02 | 0.00 |
| NEAT | R | 0.03 | 0.01 | 0.02 | 0.01 |
| NEATA | FSM | 0.02 | 0.01 | 0.01 | 0.00 |
| NEATA | SR | 0.04 | 0.01 | 0.02 | 0.00 |
| NEATA | R | 0.04 | 0.01 | 0.03 | 0.00 |

highest performance in general in terms of their WLR as noted above, and a low number of draw games compared to the other agents. The NEAT agent has a very low WR, which is caused by a very high amount of draws. This is due to its behaviour, which is similar to the SR agent, i.e. it gathers in clusters near its spawn, and waits for the opponent. Breaking the turn limit of 100 results in a high amount of draws against the SR and R agents which, in general, approach their opponent rarely. It does however defeat the FSM agent, as it is built (via its incorporated BFS method) to engage in a battle.

The XCS, XCSA and NEATA agents have a performance which was below the chosen acceptable threshold of a WLR of 50% against the three opponents, and an

7. AUTOMATED GAMEPLAY

Table 7.5: Results of the agent tournament. Each cell shows the result of 2000 games of Rock Wars in the form: column agent wins/row agent wins/draws.

| | MCTS | PF | MinMax | NEAT |
|--------|--------------|--------------|--------------|-----------|
| PF | 228/902/870 | | | |
| MinMax | 728/435/837 | 1062/145/793 | | |
| NEAT | 186/461/1353 | 501/299/1200 | 200/500/1300 | |
| XCS | 260/378/1362 | 371/435/1194 | 210/552/1238 | 62/2/1936 |

equally poor WR performance in terms of a high amount of draws games as well.

In conclusion, the MinMax, MCTS, PF and NEAT agents were determined to be adequate in various models and map combinations, thus capable of general gameplay.

After establishing a baseline performance for all agents, we set up a small tournament with agents from all main techniques (except the R, SR, and FSM agents). We were interested how well the agents would perform against each other after being trained equally. Table 7.5 shows the results of this tournament. The first observation that can be made, is that most games tend to end in a draw. With the exception of the PF vs. MinMax pair up, most of the times the agents seem unable to finish the game before the turn limit is reached. This is especially true for the NEAT vs. XCS games. This may be due to the very defensive tactics of the NEAT agent, as we will discuss in section 7.4. The other observable trend is that the PF agent clearly dominated the game tree based MinMax and MCTS agent, but achieved roughly equal performances against the XCS and NEAT agents. The other agents achieved mixed results. For a more intuitive ranking, we can order the agents by their total number of wins: PF (2836), MCTS (1402), XCS (1367), NEAT (1260), MinMax (990). The absolute numbers indicate a slightly different picture than in the test with the baseline agents. The Potential Field agent appears to an outlier due to its many wins against the MinMax agent. MCTS and XCS show equal win rates, while the XCS and NEAT agents show similar performances than before. The NEAT agent produced the highest number of “draw” games due to its very defensive tactics, basically hiding with all its units in a corner of the map, waiting for enemy attacks. A very frustrating tactic for human player, as we will discuss in the next section.

7.4 Results of human play testing

To test how enjoyable, human-like and challenging the agents were, we set up an online user test system. Through the system human participants were paired up for a *Rock Wars* game, after a short tutorial, with an agent and then for a second round with another agent. We used three different *Rock Wars* configurations, all based on findings from preceeding experiments (which will be discussed in section 9.1); players were explicitly informed about how the game’s configuration might differ between plays. Furthermore, only the FSM, MinMax, MCTS, PF and NEAT agents were used in this test. After the games a short survey was presented where players could report their preferences regarding the opponent, game and the experiment itself. The following four questions were asked after the two games were played:

1. Which opponent was more challenging?
2. Which opponent was more enjoyable to play against?
3. Which opponent played more like a human?
4. Disregarding the opponent, which game did you prefer?

All questions could be answered with either A, B, Neither or Both, where A and B refer to the first and the second game session. The numbers presented in this section are based on these self-reports.

The total number of participants was 60. The average age was 23.47 years and 95% of the participants were male. All participants played computer games in general and all participants enjoy playing strategy games. Of the participants, 45% play games for fifteen or more hours a week, 26.67% play games between ten to fifteen hours a week, 18.33% play games between six to ten hours a week and 8.33% play games between one to five hours a week. 23.33% consider themselves experts in strategy games, 55% consider themselves advanced players and 20% consider themselves novices. One participant did not answer how many hours she plays games, or what her self-assessed skill level was. It may be noted that the selection of participants is heavily biased towards young male experienced gamers, but given that most of the players were recruited in an online community for strategy games, we considered this demographic as the core audience for our experiment and this was not incorporated into our analysis; the effect of self-selection can therefore be neglected.

As can be seen in figure 7.7 and table 7.5(a), the agent with the highest win rate against humans was the NEAT agent with a win rate of 50%. The worst in terms of

7. AUTOMATED GAMEPLAY

| (a) Reported results | | | | | |
|----------------------|-------|----------|-------------|-----------|----------------|
| Agent | Games | Win rate | Challenging | Enjoyable | Human likeness |
| FSM | 28 | 21.43% | 35.71% | 53.57% | 21.43% |
| MinMax | 26 | 30.43% | 53.85% | 38.46% | 61.54% |
| MCTS | 27 | 34.62% | 33.33% | 37.04% | 25.93% |
| PF | 21 | 38.10% | 47.62% | 47.62% | 52.38% |
| NEAT | 18 | 50.00% | 55.56% | 11.11% | 27.78% |

| (b) p-Values for Human-likeness | | | | |
|---------------------------------|--------|--------|--------|------|
| | FSM | MCTS | MinMax | NEAT |
| MCTS | 0.70 | | | |
| MinMax | 0.0024 | 0.0084 | | |
| NEAT | 0.63 | 0.89 | 0.026 | |
| PF | 0.029 | 0.067 | 0.539 | 0.12 |

| (c) p-Values for Enjoyability | | | | |
|-------------------------------|-------|------|--------|------|
| | FSM | MCTS | MinMax | NEAT |
| MCTS | 0.25 | | | |
| MinMax | 0.41 | 0.73 | | |
| NEAT | 0.059 | 0.35 | 0.22 | |
| PF | 0.25 | 0.91 | 0.67 | 0.44 |

| (d) p-Values for Challenge | | | | |
|----------------------------|------|------|--------|------|
| | FSM | MCTS | MinMax | NEAT |
| MCTS | 0.85 | | | |
| MinMax | 0.18 | 0.13 | | |
| NEAT | 0.19 | 0.15 | 0.91 | |
| PF | 0.41 | 0.33 | 0.67 | 0.63 |

| (e) p-Values for win rate | | | | |
|---------------------------|-------|------|--------|------|
| | FSM | MCTS | MinMax | NEAT |
| MCTS | 0.33 | | | |
| MinMax | 0.64 | 0.61 | | |
| NEAT | 0.057 | 0.28 | 0.13 | |
| PF | 0.22 | 0.74 | 0.4 | 0.46 |

Table 7.6: Summary of human play results. p-Values $< .05$ are considered significant.

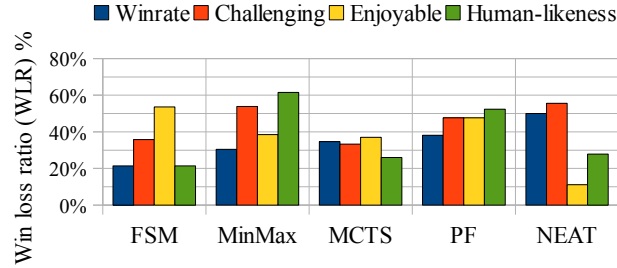


Figure 7.7: Summary of human play results

win rate was the FSM agent with a win rate of 21.43%, which was in line with the results gathered in the experiments against non-human opponents.

In terms of challenge, the NEAT agent was the most difficult with a challenge rating of 55.56%, and the least challenging was the MCTS agent with 33.33% followed by the FSM agent with 35.71%. The MinMax and PF agents are above or near 50% in terms of participants who fought the agents and found them challenging.

The most enjoyable agent was the FSM agent: 53% of the participants who fought the agent found it enjoyable. The least enjoyable opponent was the NEAT agent with only 11.11% of the participants rating it as such. The PF opponent was found the second most enjoyable agent with 47.62%, and both MinMax and MCTS were found nearly equally enjoyable.

In terms of human-likeness, the most human-like reported agent was the MinMax agent with 61.54% and the PF agent with 52.38%. The MCTS and NEAT agents were not as particularly perceived human-like with only 25.93% and 27.78% respectively. The least human-like agent was the FSM agent. P-values displayed in table 7.5(b) show that the difference in perceived human-likeness between the MinMax agent and all others but the PF opponent is significantly higher.

Although the NEAT agent was the best performing in terms of win rate and challenge provided for the participants, players reported that it was less enjoyable or human-like than the other agents. This can be explained by its passive behaviour, not approaching the enemy; some players even reported it as “broken”. This could also be observed in the tests against non-humans, where it accumulated a large number of draws against the SemiRandom and Random agents. The second best performing agent, in terms of win rate and challenge, was the PF agent. It also provided a good level of enjoyment for the participants, and was considered fairly human-like. The third best performing agent was MinMax in terms of win rate and challenge, and provided

7. AUTOMATED GAMEPLAY

a good level of enjoyment for the participants, as well as being perceived as the most human-like. The MCTS agent provided a better performance in terms of win rate and challenge than the FSM agent, and was perceived more human-like. However despite its low performance, the FSM agent was the most enjoyable agent of all five in the experiment.

Although the data is not completely conclusive, it shows that the agents based on MinMax, MCTS, Potential Fields and NEAT performed better than the simple FSM agent in terms of win rate, challenge and human-likeness. Analogously, those agents showed a superior performance in terms of win rate and win loss rate against the Random and SemiRandom agents in the non-human experiments. The only exception is the NEAT agent which was unable to approach SemiRandom and Random due to the reasons discussed. It can therefore be concluded from the non-human and human experiments that the agents based on MinMax, MCTS and Potential Fields have high skills in terms of their ability to play, that they are flexible under changing rule-sets and capable to some degree of showing human-like behaviour. Given that all agents perform a turn in less than a second for all models, we can state that all agents have shown reasonable runtime behaviour for our needs.

The NEAT agent was not enjoyable for the participants and it was not perceived human-like. It was also unable to engage the Random and SemiRandom agents. Therefore it cannot be considered as playing well in general, but has shown potential in defeating skilled opponents. The Assigner decreased the performance in agents when used to assign orders, but its ability to measure the relative power of enemy units was beneficial. The XCS, XCSA and NEATA agents have shown a poor level of skill in play against any other opponent than the Random agent, both in terms of win rate and win loss rate. The only two agent architectures that could reliably outperform all the benchmark agents (even in terms of win/loss ratio) were both tree search-based: MinMax and MCTS. This could be seen as indicating the value of game-tree search over non-searching methods. It is important to note that both the MinMax and MCTS agents used the same board evaluation function, which is a neural network trained by NEAT. (The board evaluation function was re-trained for each model.) Using the same evaluation function could explain the similar (but not identical) performance profile. Thus, the MCTS agent is just as reliant on a good evaluation function as the MinMax agent, so these results could also indicate the superiority of neuroevolutionary state evaluation functions. The NEAT agent, which learns state-action mappings rather than state-value mappings, is among the better performing agents but scores slightly lower than the tree-search agent. This finding agrees with the findings of previous

comparisons of state-value and state-action mappings in other games, such as racing games, where state-value mapping turned out to be superior (195).

7.5 Conclusions

Although the MCTS agent did not stand out in any category in the human play tests, we decided to use it in all experiments listed in the following chapters (unless stated otherwise). This decision was made for practical reasons, and due to the MCTS agent’s sufficient performance in terms of winning games and its performance in terms of computational complexity. Equipped with a meaningful board evaluator function, the MCTS agent requires no training at all to adapt to a new SGDL model. This makes it more practical to use than for example the XCS. The XCS agent requires hours of training to learn a game before being deployable. Furthermore, the MCTS agent allows us to play a large number of games in adequate time due to its flexible and controllable time-to-think parameter.

After presenting the game modelling language (SGDL) and the framework to play expressed games, we will now move on to the aspect of determining the “interestingness” of modelled games.

7. AUTOMATED GAMEPLAY

Chapter 8

Measuring Game Quality

After introducing the Strategy Games Description Language to model game mechanics, we will now discuss algorithmic game mechanics generation and evaluation. We will present several methods to quantify the player experience of a game, and therefore make assumptions about how enjoyable a game is. However, we do not differentiate player preferences yet. A further discussion about the implications of different playing styles will be presented in chapter 11.

We follow the search-based procedural content generation paradigm, established by Togelius et al. (184) among others. The term *search-based* here refers to a family of metaheuristics which start with an initial solution for a problem and iteratively try to improve it. Metaheuristics make few or no assumptions about the problem being optimised, but instead use a specifically tailored fitness function which maps a candidate solution to a numerical value. To recapitulate from section 4.1.1: search algorithms are often used whenever the number of possible solutions is simply too large for an exhaustive search for a global optimum. The vector of features that differentiate solutions is often multidimensional and the set of all possible solutions is referred to as the *solution space*, a virtual search space. On the other hand, some search algorithms tend to converge prematurely on local optima, e.g. the hill climbing algorithm (196). The family of algorithms we use in our research are *evolutionary algorithms* (EA), *genetic algorithms* to be more precise, which work on a set of possible solutions in every iteration. Inspired by biological evolution, EAs use the methods of reproduction, mutation, recombination, and selection to determine the best solution of the current iteration and the composition of the next iteration's solution set. As it is inspired by biological evolution, EAs normally refer to each iteration as a *generation*. How searching the solution space using EAs was implemented in detail is presented in section 10. This

8. MEASURING GAME QUALITY

chapter presents the three fitness function we have used in conjunction with SGDL games. This list is non-exhaustive, i.e. other approaches have been proposed and some of them have been presented in chapter 3.2. All three fitness functions are simulation based and use the artificial players presented in chapter 7. For clarity these aspects are presented separately. When the following states that “a game is played”, it actually refers to two agents simulating human gameplay.

Play Outs The following sections will use the term “play out” or “a game is played out” (also called *roll outs*). This refers to two artificial players (agents) taking alternating turns in a game modelled in SGDL; analogously to Monte-Carlo-Tree-Search (described in section 4.3.1.2). The different agents which were developed for the SGDL framework, and how they form their decisions, was presented in chapter 7, and the actual experiments and agent combinations will be described in chapter 10. But since some of our fitness functions already require playouts, an introduction seems necessary. A *play out* consist of a game and two agents. The game may be either in its initial or intermediate state, meaning that the map might already contain units. At the start of his turn, an agent receives a copy of the gamestate. He will then submit his decision and the game continues. The game ends when either one of the agents has won the game, or a stop condition of 100 turns is hit and the game ends in a draw. The latter was introduced to prevent unsolvable games being played ad infinitum without any progress, e.g. games where no unit with an attack ability is left.

Utility Functions It should also be pointed out, that all our fitness functions make use of a quantification of the gamestate or the players’ standings in one way or another. This transfers the concept of the *quantifiable outcome* (26, 110) of a game to the game’s intermediate steps, i.e. the game does not only produce a quantifiable outcome, but a hypothetical outcome is defined at all time while the game is running. Different nomenclatures exist, but we will refer to such approximation of an (hypothetical) outcome as a utility (or score) function. Utility functions are a common element in computer games. Arcade games are often designed with a utility function as a core element, and meta-games evolve around who achieves the highest score. In some cases, games are not designed to be solvable at all, i.e. the challenge for a player is to progress as far as possible, and no dedicated game end except a “kill screen” (an intentional crash of the program) exists.

However, numerical scores are rarely used as a gameplay element in strategy games. An exception is *Civilization*, but that does not imply that utility functions are not ap-

plicable. In fact, artificial agents frequently evaluate different aspects of the current gamestate to determine the best valid move. A reason why overall score functions are rarely used as a gameplay element, besides breaking immersion by making game mechanics visible to the player, might be that a reliable indicator of which player will win the game is often hard to design with low computational costs. We explored this aspect further independently from the SGDL project with the card game *Dominion* (197). Results indicated that even though the game contains a pre-designed score mechanism, other indicators were necessary to reliably predict the winner in the early- and mid-game. It also became clear, that a utility function must be tailored to a game’s mechanics and dynamics.

Finally, the function to determine the standing of a player in all of our games presented in section 6.1 made use of the health of all units on the battlefield as a score heuristic:

$$S_p = \sum_{u_p}^{U_p} health_{u_p} - \sum_{u_o}^{U_o} health_{u_o} \quad (8.1)$$

Where the score S_p of the player p is defined as the difference between the sum of health of all (U_p) his units and the sum of health of all (U_o) his enemy’s units. The assumption, that the overall sum of health of all of a player’s units relates to his chance of winning the game, is directly taken from the winning condition, to eliminate all enemy units by decreasing their health to zero. This is also an example of how a utility function fails to capture essential parts of the game mechanics. Besides the fact, that combat strength is not taken into account, the fitness ignores the economy mechanic in *Rock Wars* (section 6.1.3). Without any further context the following two states receive the same utility.

- The player just started the game and built a unit of type A
- The player is close to loosing and has just one unit of type A left

In both cases the sum of overall health would evaluate to the health of the unit of type A. However, we chose to evaluate all games with the same fitness function(s) to compare the results.

8.1 A Definition of Balance

The term “balance” is often used to describe a game’s property of permitting fair gameplay. A game is defined here as “fair” if all players have the same chance of winning regardless of their start positions, e.g. their starting point on the map, unit types at their disposal, number of units, etc. If we assume an equal skill level for each

8. MEASURING GAME QUALITY

player, every start position should provide an equal chance to win the game. This requires that the player employs the correct strategy for his start position. However, different start positions may require different strategies. Additionally, a game is defined as “balanced” if for every strategy a counter strategy exist. A “strategy” here implies a course of actions based on a decision making policy X. A “counter strategy” Y for X is therefore a strategy to ward off the effects (attacks) caused by X. Example: if player A decides to employ strategy X, which is the construction of a large air fleet to bombard player B’s base, then a possible counter strategy Y for player B would be equipping his base with enough anti-aircraft defences to fight of player A’s attack. This requires that player B is aware of strategy Y and its relation to strategy X, and he must be able to perform the actions of strategy Y at an adequate skill level.

Transferring this principle to a micro level of gameplay, unit types may also be “balanced”. Unit types are balanced if each unit type has (a) designated counter part(s), e.g. flare launchers against heat-seeking missiles. The consequence of lacking a counterpart for a unit allows a player to use it as an exploit and easily win the game. Such a unit type is often humorously referred to as a the “I win button” - often with a sarcastic undertone since such a unit type breaks the game experience and may frustrate players. Game developers often allocate a large amount of resources to balance-testing; especially for titles which are centred around a multi-player component, e.g. the closed beta testing for Blizzard’s *Starcraft II* ran over five months with 40,000 players participating (198). Naturally, beta-tests aim to tease out a multitude of aspects which would make a game unfair, but in our research we focused on the combat strength of a unit. Combat strength here refers to the utility of a unit, i.e. how strong or useful a unit is in a battle.

We start with the direct comparison of two units of different unit types: a unit type is stronger than another if it can reduce the opponent’s health faster during consecutive turns than it takes damage. Our first game “Simple Rock Paper Scissors” simulates this setup. A unit type relationship can be expressed in a directed graph. Each node represents a unit type and edges represent the comparison of two units in a fire fight: the direction of the edge indicates which unit wins. A complete directed graph specifies all the power grades. Nodes with more incoming- than outgoing edges are more likely to be weaker in the game than unit types with more outgoing edges. This method of analysis has been applied to other game types before, figure 8.1 shows an overview of the contemporary hand game “Rock Paper Scissors Lizard Spock”. Each choice has two choices it loses against, and two choices it overpowers. A simple approach to balance strategy games therefore could be to construct the unit types of a game in

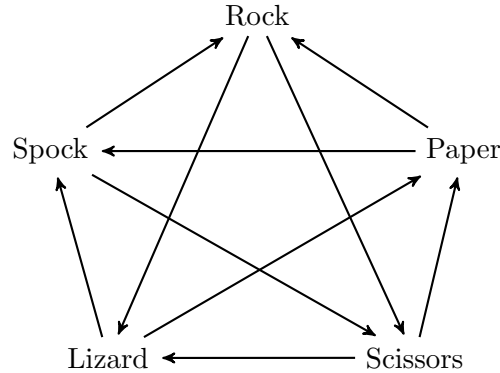


Figure 8.1: Directed graph of “Rock Paper Scissors Lizard Spock”, an extended version of the hand game “Rock Paper Scissors”. In coming edge means that option loses against the option at the other end of the edge. The creator Sam Kass writes on his webpage: “Scissors cuts Paper covers Rock crushes Lizard poisons Spock smashes Scissors decapitates Lizard eats Paper disproves Spock vaporizes Rock crushes Scissors” to explain all the combinations (199).

a way they would form a source- and sink-free graph. A source - a node which no incoming edges - would be unbeatable, and a sink - a node with no outgoing edges - would lack the ability to beat anything in the game. Additionally, another approach to balancing would be harmonising the degree of each node, i.e. each node has the same number of incoming and outgoing edges.

8.1.1 Complementary Unit Types

The concept of directed graphs has the shortcoming that power relations are binary and do not resemble the different degrees the outcome of a fight may have. It does not take the damage into account that the inferior unit does to the winning unit; the winner eventually does not come out of a fight unharmed. Furthermore, the graph does not take cooperation between units into account. This either refers two different unit types supporting each other with different abilities, e.g. “combat and healer” or “artillery and infantry”, or units which are only effective in large numbers. We follow the idea of supporting units and present a way to evolve complementary unit sets. A set of unit types is complimentary when they have different strengths, so that each of them is better than the others in some respect; and when combining units it is generally beneficial to have a balanced set rather than having the equivalent number of units of only one type. Many strategy games include the unit types which take the roles of infantry, long-ranged combatants and artillery (or something similar), where each

8. MEASURING GAME QUALITY



Figure 8.2: This screenshot from Warcraft III shows a typical fight between two parties, Orcs (red) and Humans (blue), over a building. Instead of joining the infantry in the picture’s bottom, the hero *Arthas* casts his healing spell to prevent allied soldier from dying, but deals no damage himself.

unit type has unique strengths and weaknesses so that a successful strategy depends on using them in combination. Figure 8.2 shows a common example: it is more beneficial in certain situations for a hero to act as a healer than entering the fight himself. Even though the hero is also a strong fighter in this example, a simple directed graph as described above does not take his ability to support other units into account. Another case to discuss is the intentional creation of non-combat units: real military operations consist of a supply chain with designated logistics vehicles or aircraft to provide troops with necessary supplies or transport them to the frontline. It therefore seems natural to integrate this aspect into simulated warfare. Hellwig’s Kriegsspiel (presented in section 2.1) took logistics into account, and many modern strategy games feature this aspect in one way or another (Figure 8.3), simulating the economic aspects of war. Even the economic simulations mentioned in section 2 could be seen as an extension of war.

To additionally capture this aspect, we propose a simulation-based fitness function for determining how “balanced” a game is. The following fitness was primarily



Figure 8.3: Screenshot from Blue Byte’s *Battle Isle II*. It shows a part of the game map on the left, and details of the selected unit on the right. The selected unit is an unarmed helicopter whose only purpose is to transport combat units over long distances in short times, but is not able to retaliate any hostile action.

8. MEASURING GAME QUALITY

designed for “Complex Rock Paper Scissors” and “Rock Wars” (as described in sections 6.1.2 and 6.1.3), but we believe that this approach could theoretically be transferred to other strategy games. However, the insights given by this fitness function may be insignificant or hard to obtain with a higher number of different units, as this will exponentially raise the number of required trials. We start with our simple games, where each unit type has seven relevant attributes: *health* (range $[0, 100]$), *ammunition* ($[0, 100]$), three *attack* values ($[0, 100]$) and both maximum and minimum attack range ($[0, 6]$). The attack values determine the damage that can be done by one shot on each of the enemy unit types. This means that to define a complete set of unit types, 21 values need to be specified (seven per unit class; three classes in the game).

8.1.2 Fitness function

The aim of this fitness function is to find a balanced unit configuration (here: sets of attribute values for attack, health etc.). A balanced, or complementary, set is one where the non-transitive power relations between units make it advantageous to compose armies of diverse units rather than homogeneous ones. Implicitly, we reward the cooperation between different unit types. The balance of a unit set is measured as follows: six battles are played for each unit type set. For the automated gameplay we used our MCTS agent (see section 7.2.5) and a special tailored heuristic. The very low computational complexity of this heuristic agent allowed us also to run experiments with complex scenarios in manageable time. Furthermore, it allowed us to test the effect of a very different playing style on the fitness functions, as the heuristic agent has a markedly different “temperament” to the MCTS agent. The heuristic can be described in pseudocode thus:

```
for all units do
  if unit can attack then
    attack where most damage caused (the unit against which the current unit is
    most effective)
  else if next step towards nearest enemy is not blocked then
    move towards nearest enemy
  else
    do a random move
  end if
end for
```

Balanced unit sets with units of all three types (denoted ABC) play against unbalanced sets with units of only one type (AAA, BBB and CCC). Figure 8.4 illustrates

| | | | | | | | | |
|---|--|--|--|--|--|--|--|---|
| A | | | | | | | | A |
| | | | | | | | | |
| | | | | | | | | |
| B | | | | | | | | A |
| | | | | | | | | |
| | | | | | | | | |
| C | | | | | | | | A |

| | | | | | | | | |
|---|--|--|--|--|--|--|--|---|
| A | | | | | | | | B |
| | | | | | | | | |
| | | | | | | | | |
| B | | | | | | | | B |
| | | | | | | | | |
| | | | | | | | | |
| C | | | | | | | | B |

| | | | | | | | | |
|---|--|--|--|--|--|--|--|---|
| A | | | | | | | | C |
| | | | | | | | | |
| | | | | | | | | |
| B | | | | | | | | C |
| | | | | | | | | |
| | | | | | | | | |
| C | | | | | | | | C |

Figure 8.4: Three configurations were used in CRPS for evolving complementary unit sets. The complementary unit set played against a homogeneous unit set consisting of unit types A, B, or C. All three configurations were both played with the left- and the right player starting.

the board layouts for each configuration.

To ease the effect of being the starting player, each configuration is played once with the left player starting, and once with the right player starting. Ultimately, the fitness is defined as the minimum performance achieved by the balanced set (ABC) in any of the six games. To minimize noise, the fitness calculation is averaged over 200 trials (t). The fitness can ultimately be formalised as:

$$F := \min\left(\frac{\sum^t a_1 + a_2}{t}, \min\left(\frac{\sum^t b_1 + b_2}{t}, \frac{\sum^t c_1 + c_2}{t}\right)\right) \quad (8.2)$$

where $a_1, a_2, b_1, b_2, c_1, c_2$ are defined as 1 if the player with the balanced set has won against the according non-balanced set, or 0 otherwise, and $t = 200$. More precisely: a_1 is the number of times ABC won against AAA as the starting player, and a_2 is the number of times ABC won against AAA while not being the starting player (b_1, b_2, c_1, c_2 for BBB and CCC analogously).

In “Rock Wars” the same fitness function can be used. Additionally, the cost variable for each class is added to the solution space, specifying a total of 24 values to evolve. As players start with a factory instead of pre-picked units, we transfer the concept of hetero-/homogeneous unit sets to the types of units a player could build with his factory, i.e. one factory could only produce unit type A, while the factory of player two is able to produce all three unit types (ABC); analogously for type B and C. This requires the game *Rock Wars* to be extended with a second factory class, and each player therefore starts with his own unique factory.

However, this fitness assumes that both players have access to the same unit set or a subset of said unit set. In section 8.4 we will present and discuss a generalised

8. MEASURING GAME QUALITY

version of this fitness which is capable of measuring cases where each player disposes of different unit sets.

8.2 The Outcome Uncertainty of a Game

The previous section presented a method to explore the static qualities of a game. We explored the possibility to balance a unit set using artificial agents. So far no quality assessment based on the playout style was made. Even balanced unit sets can create an uninteresting game if it is obvious to the players what the optimal strategy is. Koster (76) and other authors (200) describe this as *dominant strategies* and *inferior moves*. We already discussed units or strategies which break the game's balance in section 8.1, and a *dominant strategy* is one which is always superior regardless what the opponent plays. Many simple games can be won by a dominant strategy and are therefore considered “solved”, e.g. Tic-Tac-Toe.

We explored the phenomenon of dominant strategies and inferior choices in a different study to show that it is actually relevant for strategy games (201). We demonstrated that the utility of certain parts of the game mechanics can be quantified. Eliminating inferior choices and dominant strategies serves the goal of allowing a less predictable gameplay, without randomising the course of the game where the players' moves have no impact on the game. Our second fitness is therefore focussed more on game mechanics as a whole. Game mechanics should provide and encourage interesting games where players increase their chances by employing the right strategy rather than just pure randomness. Our assumption is that a game is interesting if it keeps a player engaged in the conflict. One aspect of an engaging conflict is its outcome, i.e. who will win or lose. Cincotti and Iida (202) theorised in 2006 (already briefly introduced in section 3.2.3), that the property of an interesting board game is high “outcome uncertainty” until a late game phase. They proposed a method to determine that uncertainty when the game is so complex i.e. has a high gametree branching factor, that an analytical determination is not possible. Instead they propose a stochastic method.

Their hypothesis makes use of the definition of quantifiable *information* by Claude Shannon (52). Shannon presented a measurement of information content associated with a random variable (a character of a finite alphabet). The term *information* here has no semantic implication and the meaning (signification) of a unit of information is completely arbitrary, e.g. the title of a book or simply the character “a” may be considered as a quantitative amount of information. Shannon introduces the notion of *entropy* as the minimum number of bits to represent a message.

8.2 The Outcome Uncertainty of a Game

Entropy is the measurement of average surprise about the outcome of a random variable, and defined by Shannon as:

$$H(X) = E(I(X)) = E(-\ln p(X)) \quad (8.3)$$

with the random variable X with possible values x_0, \dots, x_n and $E(X)$ as the expected value operator. Following equation 8.3 the entropy can be explicitly defined as:

$$H(X) = \sum_{i=1}^n p(x_i) I(x_i) = \sum_{i=1}^n p(x_i) \log_a \frac{1}{p(x_i)} = - \sum_{i=1}^n p(x_i) \log_a p(x_i) \quad (8.4)$$

Although in theory the base a of the logarithm is arbitrary, we will use the common value $a = e$ with information measured in *nat*¹ in the following. For brevity $H(X)$ can then be defined as:

$$H(X) = - \sum_{i=1}^n p(x_i) \ln p(x_i) \quad (8.5)$$

If we now consider the possible outcomes of a game as a probability distribution $P = p_0, \dots, p_k$, e.g. p_0 = probability that player one wins, p_1 = probability that player two wins, p_2 = probability that the game ends as a draw, we can also define the outcome uncertainty of any given state of a game as:

$$U(G) = - \sum_{i=1}^k p_i \ln p_i \quad (8.6)$$

with k as the number of possible outcomes, and p_i being the probability that the game will end with outcome i . Cincotti and Iida proposed the above rationale along with a method to approximate U for any given state G . They approximated the probability of each outcome of the two-player game *Synchronized Hex* using a monte-carlo simulation starting from the game state G with 1000 trials. The probabilistic distribution for state G is therefore:

$$P(G) = \frac{N_A}{1000}, \frac{N_B}{1000}, \frac{N_D}{1000} \quad (8.7)$$

where N_A, N_B, N_D are respectively the numbers where player A won the game, player B won the game, or the game ended as a draw. The more information is revealed about the run of play, the clearer the outcome of the game becomes, i.e. in a sequence of gamestates $U(G)$ normally (but not necessarily) declines. If the game permits moves such that the player can escape an unpromising position by employing the correct move,

¹*Naperian Digit* a logarithmic unit of information or entropy, based on natural logarithms and powers of e .

8. MEASURING GAME QUALITY

the outcome uncertainty might also rise. The overall tendency however will decline until it reaches zero at the end of the game.

We integrated Cincotti and Iida’s proposed method in our games with 1000 roll-outs per turn to approximate the outcome probability. In each turn, after both agents made their moves, the game state is cloned a 1000 times and each copy is played out and the game’s outcome is recorded. For each turn the outcome uncertainty defined by equation 8.7 is entered in a graph and normalised over the game’s length to the interval $[0,1]$, so that the game’s start and end equal the values 0 resp. 1. Now that the graph is normalised in both dimensions, we then used the least-square method to create a fitting over the actual uncertainty graph. This was done to efficiently calculate the distance d . Following the stipulation that the final loss of uncertainty (final game phase) should occur as late in the game as possible, we numerically computed the distance d of the nearest point on the curve close to the point 1.0, 1.0 (the maximum uncertainty, game length). Since we try to minimize this distance, the resulting fitness function is $1 - d$.

8.3 Avoiding a Start-Finish victory

The previously described way to approximate the outcome uncertainty of a game, and our application to strategy games, is very robust regarding required expert knowledge about the game. Besides the different outcomes of each game no further assumptions are made by the fitness function. However, the computational cost of 1000 simulations per turn make it infeasible for longer games with more gameplay mechanics (resulting in a larger search space of valid move options). Yet, we want to focus on another criterion of strategy games gameplay which is related: the thrill of a game, when players have sudden sensations of excitement. Even though strategy games are often slow paced, single actions or events often cause the spontaneous outbreak of emotions of joy or anger on an intermediate triumph or setback. Our general assumption is that a game is more interesting the more thrilling it is. It should be clarified, that “thrill” may be a necessary aspect, but is not sufficient to create an interesting game. It is debatable if a very abstract and random game is interesting, even though it might create the sensation of surprise when players discover unanticipated effects of their actions. Further investigations about the correlations with other aspects, e.g. can a game be learned, how well can gain a skilled player an advantage over a novice one, would clearly be interesting.

However, one method which seems to approximate the aspect of “thrill” very well is the “lead changes” indicator used by Browne for combinatorial games (95) (presented

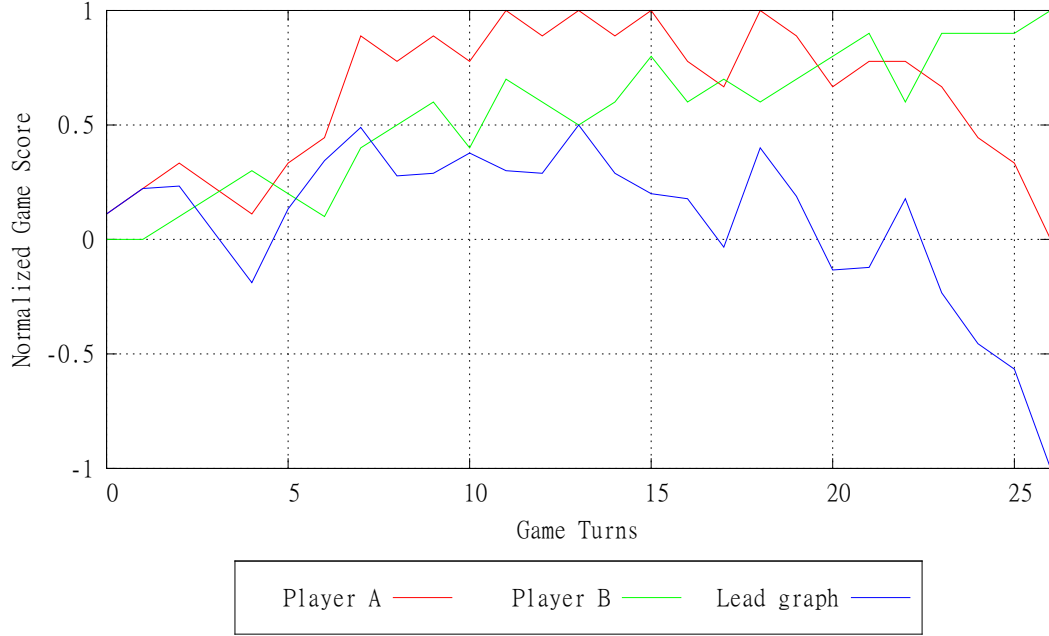


Figure 8.5: The normalized scores (red and green) for two players of an exemplary game. The lead graph (blue) indicates the current leading player.

in section 3.2.4). *Lead change* herein refers to a utility function which assigns each player a numerical value at each step in the game. The value is proportional to the player's standing in the game, i.e. a higher chance to win the game results in a higher utility. If the player with the highest utility is a different player than in the last turn, and therefore a new player is leading the scores resp. the game, a *lead change* has occurred. Assuming that a utility function to determine the leading player exists, the scores for each player is recorded for every player through every step of the game and normalised to the interval $[0, 1]$, whereas 1 represents the maximal score recorded. A third graph (the lead graph) is created representing the difference between the two players' scores. Given that the two score graphs range between 0 and 1, the lead graph can only range between -1 and 1 . If the first player leading the game, the lead graph will have a positive data point in that turn, a negative data point if the second player leads. The example in figure 8.5 illustrates this more clearly: red and green are each player's performance graph normalized to the maximum score. Blue represents the difference between the score graphs. Every time the sign of the lead graph changes, the leading player changes. Following our assumption, that a larger number of lead changes indicate a more interesting gameplay, our third fitness function optimises for the average

8. MEASURING GAME QUALITY


number of lead changes per game. The actual fitness is therefore the number of lead changes divided by the number of turns the game lasted. Although this fitness does not explicitly prevent inferior-choices, dominant-strategies or players' action having no effect, these flaws will not abet a high fitness score. If one player follows an optimal strategy, which by definition renders the opponent's strategy insignificant, and therefore will lead the game from the game start to its end, no lead change will occur. Ergo the resulting fitness is zero. The opposite case, where one player is playing an inferior strategy, or chooses action with no effect, and the other player is playing an arbitrary strategy, will lead to the same result.

8.4 Fitness functions for Action subtrees

This section discusses fitness functions for the evolution of subtrees of Actions to create unit types with new behaviour. The actual experiment will be described in section 9.2.1. Here, we present a generalisation of our "Balance" fitness function, and use an asymmetry measure for SGDL trees as an indicator of how distinct two unit sets are. We further use an auxiliary fitness functions "Maximum Damage" to test the behaviour of our system.

The three used fitness functions operate on well-defined SGDL trees. SGDL does not guarantee semantically correct trees, but allows to detect syntactical faults, e.g. referencing non-existing attributes. Although model integrity checking is a well-known problem discussed in software engineering research, we decided to keep our system simple. However, whenever an exception during the simulation phases due to an ill-defined SGDL tree occurs the candidate solution is assigned a score zero. It does however stay in the population to ensure diversity. Two of our fitness functions here are simulation based like our previously described fitness functions, i.e. agents play games with the rules evolved against each other. But to improve the selection process and save computation time, we implemented a few pre-sampling steps to prevent the system spending time on unpromising genotypes. As the winning conditions are static for all the games (to eliminate the enemy units until he has no units left nor can he buy new ones), we know that each SGDL model has to contain at least one action that reduces the health attribute of a target. With this knowledge we can traverse the tree, look for a subtraction-operator node and test if its children are a constant attribute with a positive integer number, and a property attribute node referencing a "health" attribute of a target. If no such node can be found, the phenotype won't be tested through simulation and assigned a score of zero. Even with large and complex

tress, traversing the nodes is significantly faster than simulating a game using it. An exception forms the Asymmetry fitness which is not simulation based, but instead a deterministic value is calculated; as this is very light computationally, no pre-checking is done. The following three fitnesses were used to evolve Action subtrees:

Maximum damage As a simple test of system we designed a fitness function that recorded all changes of the attribute “health” of any unit in the game and simply summed up the amount of damage done as a fitness value. The applicability for game design might be limited, although generating unit sets with a huge amount of impact might be a use case, it allowed us to verify that our framework evolved SGDL trees successfully. This fitness function differentiates between unit sets, it therefore may be possible to create a unit set that completely dominates the opposing unit set. It is important to note here, that for this fitness function the pre-sampling step of traversing the tree for a -node referring to a health attribute was disabled. Although not strictly necessary, we disabled this step to not restrict the fitness function and create a skewed fitness landscape. All solutions that would pass the pre-sampling test would naturally achieve a high score in the “maximum damage” fitness.

Asymmetry The aspect of fairness plays an important role in multiplayer games. Fairness here refers to the equality of each player’s chance to win the game. Most games have as an ideal that a player’s chance of winning the game only depends on his experience and skill with a game, and only little on external factors such as starting positions or rules. Some games offer the option of handicaps, where a player who is more experienced may chose a malus in order to keep the game interesting. Symmetry plays a significant role in multiplayer gaming with levels or maps to provide the most fair competition: team based first person shooter maps for competitive gaming are often symmetrical, e.g. teams have equally shaped bases or similar. This also seems to be the case for strategy games according to findings by Togelius et al. for generated maps for Starcraft (120). On the other hand, *StarCraft* and *Civilization*, two of the most successful commercial strategy games, are popular for their use of asymmetric unit sets. The races in StarCraft differ so significantly from each other in a way that they require different strategies and playing styles to win the game. Most professional eSports players are therefore specialised in the race they are normally playing. Specialisations in Civilization are a bit more subtle, selecting a different nations will give the player a different set of perks, endorsing a slightly different playing style, e.g. culture over military victory.

8. MEASURING GAME QUALITY

The unit types available to a particular player can be summarised as a “unit set”. If two unit sets are significantly different from another, we will refer to them as “asymmetric”. Our second fitness function tries to formalise this, and evolve SGDL trees that maximise this aspect. To quantify how different two unit sets are, we take advantage of the fact that they are both defined as trees. A survey of tree difference measures can be found in (203). We will use the *constraint edit distance*: given two trees T_1 and T_2 , the distance measure is defined as the total cost of deletion, insertion and edit operations we have to apply on T_1 ’s nodes to transform it into T_2 . Although we can apply this to a SGDL trees immediately, the following example shall illustrate what problem may occur: define an arbitrary unit type u and a tree T_1 which contains u . Now define another tree T_2 which also contains u but also an exact copy of u , named u' . The edit cost between T_1 and T_2 would be either deleting u or u' while in fact both subtrees provide the same options to the player, i.e. there is no relevant difference in the game mechanics. But if u' would only marginally differ from u , the problem would still exist. To solve this we introduce a way of determining which child of a node to delete and what its deletion cost is. Define trees T_1 and T_2 , both having child classes C_0, \dots, C_K , each a set of children c_{k0}, \dots, c_{kN} . To determine the distance between these nodes we first find for every class k the node t with the most children and t_0 with the least children. If the amount of children in class k is equal, we skip this first step. We then pick the following set S for deletion, where $|S|$ is the difference in amount of children.

$$\operatorname{argmin}_S \gamma(M(C_k^t \setminus S_{ki}, C_k^{t'})) \quad (8.8)$$

Where $M(A, B)$ is the mapping between nodes in A and B and γ its cost, determined by summing the distances between nodes mapped to each other. The distance between T_1 and T_2 now is the average of the costs in M plus the minimum distance of every $s \in S$ to any of its siblings in the same class:

$$\sum_k \gamma(M(A, B)) / |M| + \sum_s \min_i \delta(c_{ks}, c_{ki}) \quad (8.9)$$

This ensures that a different number of unit classes, actions, attributes, i.e. a different number of tree nodes, contribute proportionally to the distance measure, depending on how different it is to its siblings. We also average the cost because we don’t want to increase the distance measure just because there are children to measure in between. This makes the distance measure depend on the branching factor, and ensures that maximising the distance measure does not just maximise the number of child nodes.

To measure the asymmetry of the SGDL tree of an individual, we virtually divide each tree into the unit classes assigned to player one and those assigned to player two. Each unit set is combined under a virtual root node, and the constrained edit distance between the two unit set is the overall score for an individual for this fitness.

Balancing This fitness function tries to approximate the “balance” of two unit sets. With the balancing-fitness as described in section 8.1 we tried to approximate balancing using symmetric unit sets by only evolving the parameters of the attack-actions. As the actual behaviour and the number of classes were static, we could use a priori knowledge to test how balanced two unit sets were: each unit set contained three units, if a unit set of only one type of each unit could win against a heterogeneous unit set (composed of all three unit types), a unit set was considered as unbalanced as it contained a “killer unit”, i.e. a player could win the game by just using that unit type.

In the experiments with a dynamic number of classes and actions, no a priori information is available: the number of unit classes and actions are determined through the evolutionary process. Only the attributes and winning conditions are static. The evaluation through this fitness is a two-step process:

1. The game defined by an individual is played by two agents that use a random-selection policy to determine their actions. In each round they pick a (valid) action to play. If one of the random agents is able to win a game before the limit of 100 rounds is reached, the individual will not be evaluated further and is assigned a fitness of zero by this fitness. The background is, that a game which can be won by chance is probably uninteresting, and does not comply with the definition, that a strategy game requires “strategic thinking” (27).
2. Two Monte-Carlo Tree search (MCTS) agents play the game using the evaluator function that previously has been developed for our *Rock Wars* game (47). This is possible as the attributes and winning conditions remain the same, i.e. the goal of the game is still to reduce the health of enemy units. Using our re-sampling factor, 200 games are played using the MCTS agents. The theoretical optimum is achieved when both players have a chance to win the game of 50%. We therefore take the minimum of the win rates of both players, normalise it to the interval of $[0, 1]$, and use it as a score for the individual by this fitness.

8. MEASURING GAME QUALITY

Chapter 9

Searching the Strategy Game Space

After having presented several methods to approximate the quality of game mechanics of a strategy game, we would like to present how these methods were used. We conducted several experiments with our example games from section 6.1. Following the search-based paradigm for procedural content generation, the approach for all experiments is similar: an evolutionary algorithm evolves a population of potential best candidates for a fitness function. The next generation is formed using operators such as reproduction, recombination, and mutation based on fitness of the solutions in the current generation. While the previous chapter focussed on how to assign a score to a certain individual, this chapter focusses on how to represent strategy games for certain fitnesses and purposes, which algorithms we used, and which results were produced by our experiments. We start with the evolution of simple parameter sets, compare the results of different fitness values, and conclude with the evolution of complete subtrees of Actions. It could be argued, that the newly created actions are an extension of the human game designer's creativity.

It should be added, that SGDL was also created to help human game designers in developing and prototyping strategy games. As literature concerning mixed-initiate approaches shows, generative algorithms can be used to extend, augment, or even replace parts of the design process. With that in mind we created a simple application to visualise and edit SGDL game trees (figure 9.1) for our own purposes. It allows the basic creation of trees, but is not feasible for lay persons to create games without assistance. The software's user interface has shortcomings in its current stage, and the development effort needed to achieve a certain state of usability, lead us to the decision

9. SEARCHING THE STRATEGY GAME SPACE

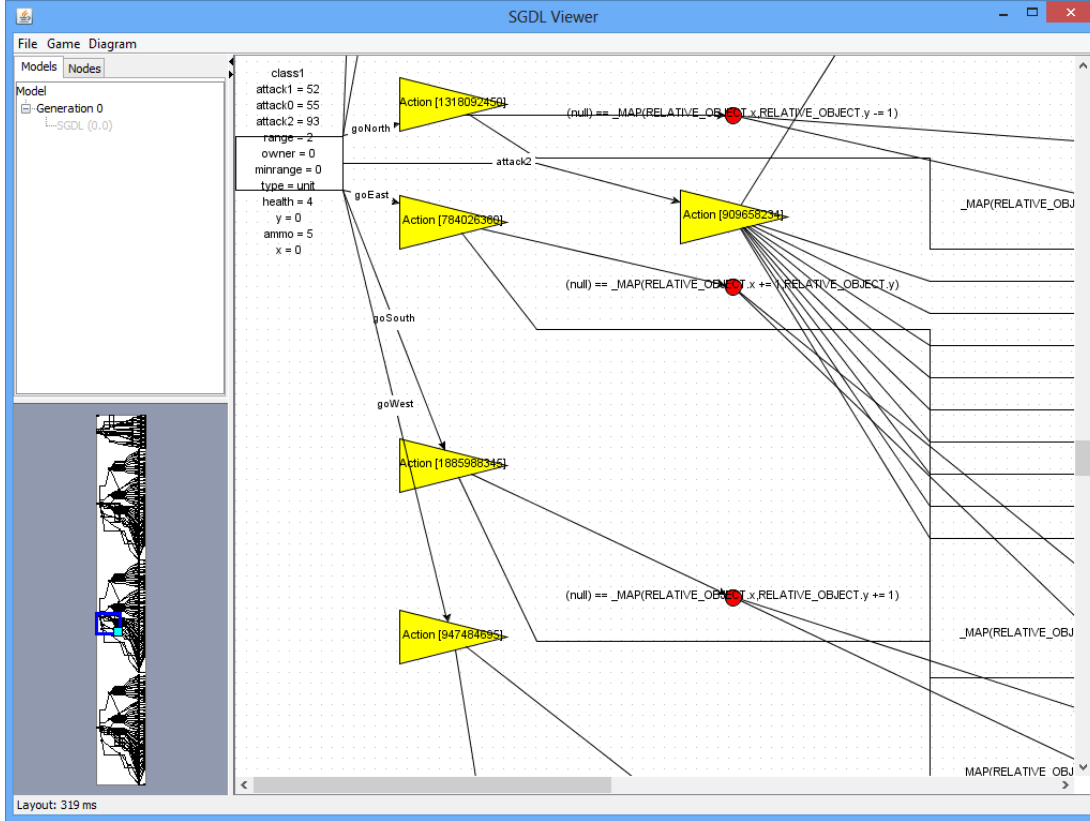


Figure 9.1: Screenshot of the SGDL Editor prototype. The main portion of the screen shows an ObjectClass with several actions as child nodes. The lower left part shows the overview of the complete SGDL model (without the SGDL root node). Each tree structure relates to an ObjectClass. The upper left part of the screen can be used to navigate through several results of the evolution process (only one in this example). The editor can also be used to launch a SGDL model in the game engine.

of halting the development of the toolkit and focus on the planned development of the *evolution component*. The evolution component is designed to evaluate SGDL trees based on the fitnesses presented in chapter 8 and evolve and improve existing, humanly designed, or randomly generated game mechanics; which would make them more interesting or spot flaws such as inferior choices or dominant strategies.

9.1 Balance and Complementary Unit Sets

The first experiment targets complementary unit sets (as defined in section 8.1.1) using our “balance”-fitness described in section 8.1 and the *Complex Rock Paper Scissors*

9.1 Balance and Complementary Unit Sets

Table 9.1: A unit type set with fitness 0.0.

| Type | Health | Ammo | Attack 1 | Attack 2 | Attack 3 | Min range | Max range |
|------|--------|------|----------|----------|----------|-----------|------------|
| A | 53.0 | 33.0 | 60.0 | 20.0 | 92.0 | 10.0 | 0.0 |
| B | 82.0 | 78.0 | 85.0 | 60.0 | 62.0 | 0.0 | 23.0 |
| C | 39.0 | 45.0 | 37.0 | 100.0 | 12.0 | 0.0 | 0.0 |

game described in section 6.1.2. We implemented a standard genetic algorithm using a 21-dimensional genotype: Each unit type has seven attributes: *health* (range $[0,100]$), *ammunition* ($[0,100]$), three *attack* values ($[0,100]$) and both maximum and minimum attack *range* ($[0,6]$). The attack values determine the damage that can be done by one shot on each of the enemy unit types. We employed a $\mu + \lambda$ evolution strategy with $\mu = \lambda = 50$ (elitism, keeping the top 50% and replacing the others with offspring of the top 50%). For simplicity, neither crossover nor self-adaptation was used. The mutation operator added Gaussian noise with $\mu = 0, \sigma = 0.1$ to all values in the genome. The gene values were normalized to real values in the range $[0, 1]$. The games were played by our Monte-Carlo agent described in section 7.2.5. To eliminate the effect of noise, each fitness calculation was averaged over 200 trials.

A key research question in this experiment was whether the fitness function accurately captures the desired property of complementarity, and whether the highly fit unit type sets are more interesting to play than poorly fit sets. To shed some light on this, we analysed a few evolved unit type sets which were published in 2011 (204). Table 9.1 presents one unit type set with fitness of 0.0. We can see that that this particular set contains two basically non-functional unit types: the A and C unit types are unable to shoot given that their shooting range is zero. While games against AAA and CCC will always end in favour of ABC, ABC will never win against BBB. Even though ABC contains one functional unit and may even kill one unit of BBB, it will always be eliminated by the second unit of BBB. Therefore there exists a dominant combination that always wins over all other combinations, making this configuration very uninteresting to play. Table 9.2 presents a set with fitness of 0.24, which is a mediocre score. While all three unit types appear to be functional and have different strengths and weaknesses, this configuration does not perform very well. We observe that all three types have very similar minimum and maximum ranges. In conjunction with the alternating turn order it may become a losing proposition to ever engage an enemy unit. The unit that moves in range first will inevitably be the first one to take damage since the enemy moves next. As our MCTS-based player will avoid most such

9. SEARCHING THE STRATEGY GAME SPACE

Table 9.2: A unit type set with fitness 0.24.

| Type | Health | Ammo | Attack 1 | Attack 2 | Attack 3 | Min range | Max range |
|------|--------|------|----------|----------|----------|-----------|-----------|
| A | 46.0 | 69.0 | 61.0 | 71.0 | 71.0 | 2.0 | 5.0 |
| B | 6.0 | 43.0 | 22.0 | 90.0 | 22.0 | 3.0 | 5.0 |
| C | 36.0 | 82.0 | 40.0 | 47.0 | 6.0 | 2.0 | 4.0 |

Table 9.3: A unit type set with fitness 0.57.

| Type | Health | Ammo | Attack 1 | Attack 2 | Attack 3 | Min range | Max range |
|------|--------|------|----------|----------|----------|-----------|-----------|
| A | 6.0 | 82.0 | 39.0 | 2.0 | 67.0 | 0.0 | 3.0 |
| B | 4.0 | 31.0 | 92.0 | 79.0 | 3.0 | 1.0 | 5.0 |
| C | 64.0 | 79.0 | 94.0 | 1.0 | 90.0 | 0.0 | 2.0 |

moves, most games will be counted as unplayable after a turn limit of 100. The positive fitness is probably because some games are won by one party or another by pure chance. Table 9.3 presents the top-scoring individual found during one evolutionary run. The unit types' attack values are rather complementary — each unit type vulnerable against at least another type. We see also see that type C has more health than the other types. Type A and B can be seen as support units, while type C is more of a general purpose combat unit. Units of type A and B can be killed with a single shot.

Analysing the model further, we see that range and ammunition have no effect on the gameplay. Since the evolutionary algorithm had no constraint or heuristic of evaluating the attributes, these values became quite large. Ammunition values are such that all units may shoot without any shortage of ammo which makes this game element practically irrelevant. A human game designer might now decide to remove this mechanic completely, manually edit the values or add constraints to the evolutionary algorithm. If we recreate this scenario in our simulator, we observe that a round often ends after a series of one-hits, meaning a unit kills another unit with one shot. There is no spatial movement necessary anymore. Nevertheless, it turns out that the balanced set in fact wins over the unbalanced set most of the time. While this genome seems to score high with our fitness function, we doubt that this unit configuration would appeal to players at all. The generated solution reduces the gameplay to a few simple decisions on which unit to shoot and in what sequence. It might be also possible, that the *Complex Rock Paper Scissors* (CRPS, section 6.1.2) game used in this experiment is too simple to support interesting gameplay at all. The main critique of this experiment is that the used CRPS' rules are too simple, and the map layout was too plain.

9.2 Comparing balance to outcome uncertainty

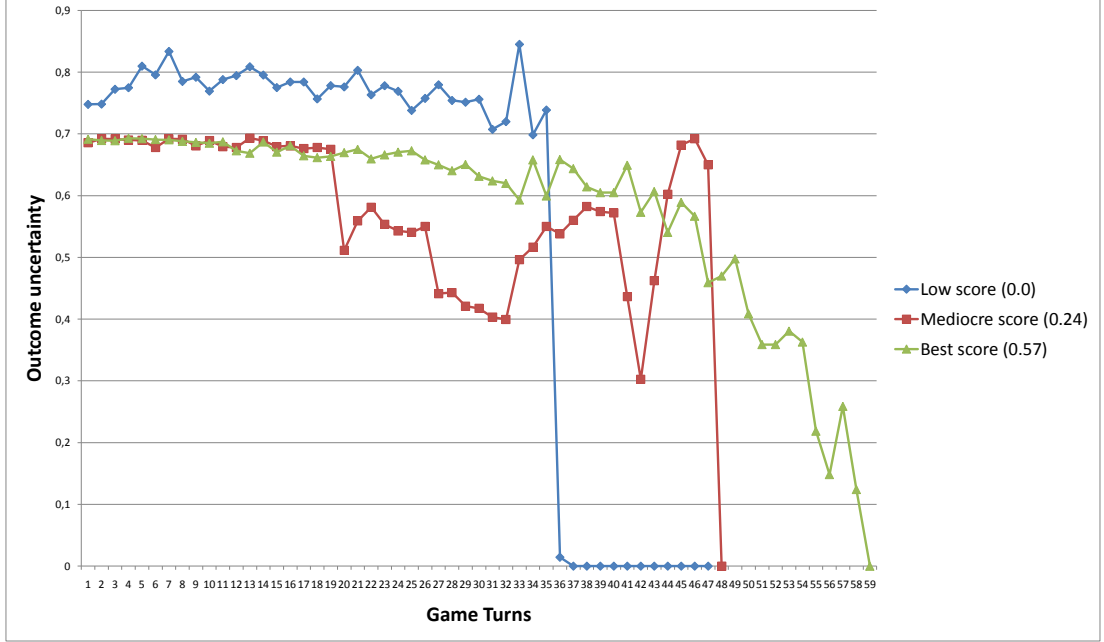


Figure 9.2: The outcome uncertainty in *Rock Wars*. The configurations were based on our CRPS configurations: the blue graph (“low score”) is based on the values in table 9.1, the red graph (“mediocre score”) is based on table 9.2, and the green graph (“best score”) based on values in table 9.3.

9.2 Comparing balance to outcome uncertainty

The logical next step was to expand the game into *Rock Wars*, adding more elements that strategy games normally contain, and validating the used fitness with additional measures. To increase the complexity of the game and make it more appealing to human players, we increased the map size to 20×30 and randomly placed impassable rock objects on 10% of the map tiles. Strictly speaking, we would need to check if the non-deterministic creation of the scenario does not create unplayable games (even if the configuration is valid) e.g. through creating impassable barriers, but in practice non-playable maps have never been observed. To compare the findings to the previous experiment, we introduced two new fitnesses: outcome uncertainty and lead changes, as described in sections 8.2 and 8.3. For a initial comparison between *Complex Rock Paper Scissors* and *Rock Wars*, we created several *Rock Wars* configurations based on our CRPS findings presented in tables 9.1, 9.2, and 9.3 and evaluated them. Figure 9.2 shows an example of the development of the outcome uncertainty for our three different configurations. The blue graph (“low score”) is based on the values in table 9.1, the

9. SEARCHING THE STRATEGY GAME SPACE

red graph (“mediocre score”) is based on table 9.2, and the green graph (“best score”) based on the values in table 9.3. For the sake of readability we will refer to the graphs by their colour.

We see that each configuration generates its own unique graph. All three graphs show a very stable first part until the first shots are fired. The *Blue* graph shows that the game is decided immediately once a shot is fired; hence the uncertainty drops to the base level of 0. That there are several turns more needed to end the game could have one of several explanations: either the game requires that the remaining units have to be simply cleared from the battle field to trigger the win condition. In the first *Command & Conquer* game a player had to clear all opponent’s passive structures like walls and sandbags to win a multi-player match. Even if the opponent was unable to fight back at all since he had already lost his production facilities and units. Another possibility could be the existence of an obvious action (killer- or finishing-move) that an agent has to take but due to its non-deterministic nature does not pick. *Red* shows a slight decrease of the uncertainty in the beginning until the graph starts oscillating heavily. We consider *Blue* merely a pathological case: the configuration is basically non-functional since two unit types have a range of zero. But compared to that, we simply consider *Red* a “bad” configuration, and *Green* shows a graph we would suspect from a well working configuration. It is surprising here that the fittest configuration from our previous experiment shows the slowest decrease of uncertainty in this experiment. Following our measurement of the smallest distance to the point 1.0,1.0, this fitness is rather low (0.5) compared to the mediocre (0.38) and bad configurations (0.46).

To gain further insight about all fitnesses, we sampled hundreds of both scenarios. Because the development of the MCTS agent (described in section 7.2.5) was not finalised at the time the experiments were conducted, the majority of the games were played by an agent which used a simple heuristic (as described in section 8.1.2) which was abandoned later in the project, that played very aggressively but with some rudimentary target selection. For each combination of scenario and agent, all three different fitness measures were calculated, and the correlation between the different fitness measures calculated. We tried to aggregate as much data as possible for each experiment in a reasonable time, whereas different computational costs lead to a different number of simulation results for each agent/game combination: 7000 for *Complex Rock Paper Scissors* using the heuristic and 900 using the Monte-Carlo agent. For *Rock Wars* we sampled 675 games using our heuristic. Unfortunately we were not able to simulate a significant number of Rock Wars games using our Monte-Carlo agent when we published the data in 2011 (205). Looking at the data presented in tables 9.4 and 9.5

9.2 Comparing balance to outcome uncertainty

Table 9.4: The correlation between different properties of sampled games of CRPS. 7000 games were sampled using the heuristic (a) and 900 using the MCTS agent (b). Significant values with gray background.

| (a) | | | |
|--------------|-----------|--------------|-------|
| | Balancing | Lead changes | Turns |
| Uncertainty | 0.04 | −0.08 | −0.12 |
| Balancing | | 0.01 | −0.1 |
| Lead changes | | | −0.17 |

| (b) | | | |
|--------------|-----------|--------------|-------|
| | Balancing | Lead changes | Turns |
| Uncertainty | 0.17 | 0.09 | −0.16 |
| Balancing | | 0.19 | −0.36 |
| Lead changes | | | −0.53 |

Table 9.5: The correlation between different properties of ca. 675 sampled games of *Rock Wars* using the heuristic. Significant values with gray background.

| | Balancing | Lead changes | Turns |
|--------------|-----------|--------------|-------|
| Uncertainty | 0.01 | 0.22 | 0.03 |
| Balancing | | 0.02 | −0.03 |
| Lead changes | | | −0.19 |

we observed several interesting correlations. Some of them can be discarded as non-significant. Correlations are given as the Pearson correlation coefficient (r), while their significance was tested against the null hypothesis ($> 95\%$ confidence):

$$t = \sqrt{\frac{n-2}{1-r^2}}$$

whereas r is the correlation coefficient and n the number of games played. Although these are rather small, we would like to focus on three details here: A) the correlation between the game lengths and the number of lead changes shows a significant correlation in both games. This is rather expected as we assume that the rate of lead changes is a constant property of the game mechanics. It is only natural that the total percentage of lead changes decreases with an increasing game length. Therefore a negative correlation can be observed. B) There is a weak correlation between the uncertainty measurement and the number of lead changes in *Rock Wars*, while in *Complex Rock Paper Scissors* there is not. While a correlation between tension throughout the game is to be expected

9. SEARCHING THE STRATEGY GAME SPACE

to depend on the number of lead changes, we believe that we don't see any correlation in *CRPS* as it is basically too simplistic to be able to create tension and there are not many lead changes due to the short game length. We discard the negative correlations of -0.08 (Heuristic) and 0.09 (MCTS) as non-significant. C) The simple CRPS scenario shows a significant negative correlation between the uncertainty and the game's length ($-0.12/-0.16$). We assume that a game is either over very fast, or it is in a non-functional configuration where the game end is only prolonged due to the fact that no party can win (e.g. only having units with a range of zero left). While the values for the Lead changes and uncertainty behave very similarly between the two agents for *CRPS*, we observe several differences in the other correlations, especially in those that base on the game length. We believe that this is connected to the increased average game length: 20 turns (heuristic) versus 60 turns (MCTS). The MCTS agent is certainly more versatile than the heuristic, which is hand-crafted to work with the current scenarios, but has a markedly different and more cautious playing style. Note that there is no straightforward relation to playing style.

9.2.1 Evolving actions

After successfully evolving attribute values within the SGDL tree, we advanced to evolving complete subtrees. The aim is to not just evolve characteristics of behaviour, but evolve the behaviour itself. For this purpose we designed another genetic algorithm which genotype was the SGDL tree itself (as opposed to a set of attributes before). Selection, procreation, and mutation is done through operators known as genetic programming. The fitness of each solution is determined using three fitness functions, whereas two are simulation based, i.e. a game defined by an SGDL tree is defined by an artificial agent while different measures are recorded. Additionally, we present a evaluation function which operates on the SGDL tree itself.

The game which forms the context for all individuals in our experiments remains *Rock Wars*. We removed all ObjectClasses from the game model but rocks and the factories. However, the genetic algorithm was somewhat restricted, as each unit class kept several attributes that determine the movement speed, and several combat related features. The abilities of rocks and factory remain fixed (rocks have no abilities, factory has a "create" actions for each unit type it can produce. Each unit class has an "owner" attribute which controls which player may use it. The abilities of the unit classes are generated through the evolutionary algorithm, but each unit at least posses the basic movement abilities such as *goNorth*, *goSouth*, *goEast*, and *goWest* which will move a unit in the respective direction if the target position is free, i.e. no other object

9.2 Comparing balance to outcome uncertainty

Table 9.6: Overview of the attributes of each class. The “Unbound attributes” refer to attributes which were formerly used in statically defined attack functions. Now they may be freely referenced like the other attributes.

| Class | Attribute | Description |
|------------------------------|-----------|---|
| Rock | x | Horizontal coordinate |
| | y | Vertical coordinate |
| Factory | x | Horizontal coordinate |
| | y | Vertical coordinate |
| Units | owner | Player who owns the building |
| | x | Horizontal coordinate |
| | y | Vertical coordinate |
| | health | if health drops to zero, the unit is deleted |
| | speed | how many tiles a unit may move per turn |
| | owner | which player may create this unit in the game |
| | cost | cost to create a unit |
| Unbound attributes for Units | | |
| | minrange | a minimum range |
| | maxrange | a maximum range |
| | ammo | ammunition capacity |

is residing there. Although important to note is, that we implement an observation mechanism that removed a unit from the game board once its health reaches zero or lower. The original model implemented the “remove from map” actions through the SGDL tree, but initial test showed that the evolutionary algorithm produced too many non-winnable games unless it produced a “remove from map” construct on very rare occasions.

9.2.2 The solution representation

A solution is a SGDL tree containing several ObjectClass-nodes with individual action subtrees. Each action-subtree consists of several condition- and consequence subtrees which again consist of several subnodes, referencing object attributes and constant values. The number and names of the attributes of each class are pre-defined (please refer to the previous section). The “owner” attribute of each class is used in the evaluation process to assign classes to a certain agent, i.e. the SGDL tree defines which player may use and produce a certain unit type. Solutions where one player may choose

9. SEARCHING THE STRATEGY GAME SPACE

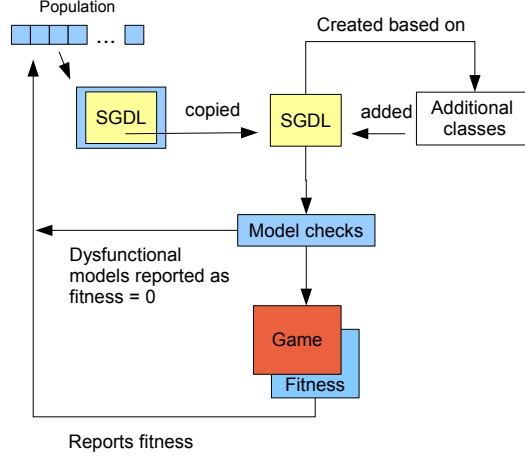


Figure 9.3: Overview of the evolutionary algorithm

between several classes to build and another player has to rely on one are explicitly allowed.

9.2.3 The Evolutionary Algorithm

The evolutionary algorithm used in this experiment used a population size of 25 individuals and each run was done over 50 generations. We used a crossover rate of 0.9 and a mutation rate of 0.05. Selection was done via tournament selection with a tournament size of 5. The crossover operator was implemented as following: two individuals are selected, and a random node within the first individual is selected. The second step is to pick a random node within the second individual that has the same type as the randomly selected node in the first individual. In the last step the two nodes and their subtrees are swapped between individuals. The mutation operator implemented, that a random node was picked, its child tree deleted and randomly regrown. The simulation based fitnesses used a sample factor of 200, i.e. each solution was played 200 times with the overall average as the resulting fitness. Our preliminary experiments showed a significant noise in the fitness function, which could be successfully dampened through re-sampling, hence the sampling factor of 200 was chosen.

To reduce the complexity of the experiment, we did not include all the classes necessary to play *Rock Wars* in the genotype. Instead whenever a solution is tested, a copy is used with the classes rock and both factories added procedurally. Each factory class (one for each player) is given a *create* action for each unit class the corresponding player may use. The resulting SGDL tree is then used to determine the candidate

9.2 Comparing balance to outcome uncertainty


solution fitness. For our simulation based fitnesses an instance of game specified by the SGDL tree is created, and played by two artificial players. While playing the game the SGDL framework feeds its events into one of the fitness functions. Figure 9.3 shows an overview of the overall process.

After introducing the experiment setup, we will now discuss the obtained results in the next chapter.

9. SEARCHING THE STRATEGY GAME SPACE

Chapter 10

Overall Results

After we established the experiment setup in the previous chapter, we will discuss the results of the evolutionary algorithm, using the fitness functions presented in section 8.4. The initial experiment, using the “Maximum damage” fitness was a first functional test for our system. The analysis of the highest scoring individual showed the expected occurrences of several -nodes referring of health attributes, i.e. several actions decreased the health of one object or another. A further analysis showed that the used operands grew over each generation. We stopped the experiment after ten generations as there is no theoretical limit as the genetic operators can just increase the operands almost ad infinitum.

A similar problem occurred with the *Asymmetry* fitness: our tree-distance fitness counts the operation necessary to transform one tree into another. With no upper limit in the size of the tree there is also no theoretical limit in the tree distance. Normalising the tree-distance to the interval $[0,1]$ is therefore only possible when all solutions of all generations have been evaluated. But since our experiment required combining the *Asymmetry* fitness with our *Balancing* fitness, we limited the total number of nodes within a sub-tree and divided the score of the *Asymmetry* fitness by a large enough constant.

Our main experiment with the two different fitness functions used a linear combination of both functions as an overall score for each individual. Different weights were used in five different experiments to study the effect of each function, isolated and in combination with the other. The following paragraphs report the findings of each of the five experiment run, sorted by weight-configuration. An overview of all the weights used can be seen in table 10.1. The following experiments were each carried out of several times to gain a idea of the stability of the results. However, as each single

10. OVERALL RESULTS

Table 10.1: The weighting factors used in five different experiments.

| γ (Asymmetry) | λ (Balance) | Comment |
|----------------------|---------------------|------------------------------------|
| 1.0 | 0.0 | Only Asymmetry |
| 0.75 | 0.25 | Combined fitness towards asymmetry |
| 0.5 | 0.5 | Equally weighted fitness |
| 0.25 | 0.75 | Combined fitness towards Balancing |
| 0.0 | 1.0 | Only Balance |

run took up to several days to run, we were unable to create enough data to create a statistical analysis of the results until the time of writing. We therefore present the results in the form of a qualitative analysis.

Only Asymmetry ($\gamma = 1.0, \lambda = 0.0$) The experiment using only the Asymmetry fitness turned out to be the hardest to analyse: as there is no simulation done, we have no a priori information if the game is actually winnable even with the highest scoring individual. A visual inspection of the tree showed a huge difference of classes available to player one and two (one versus three). A closer analysis of the ObjectClass of player one showed that winning as player one would be impossible: the only usable action had the subtree as seen in figure 10.1. Even though units of that class available could attack other objects, they would sacrifice themselves in doing so. Therefore the game could only be won as player two, who had unit classes at his disposal which featured a straight forward “kill” action. Further analysis of the complete individual brought as to the conclusion that the high asymmetry score was caused, besides the uneven number of classes available to the players, by an uneven numbers of actions available to each unit, and a high percentage of actions which adjusted different attributes than health, i.e. while player one’s units could only suicide attack, and one class of player two’s unit could properly attack, were the rest of player two’s units rather unconventional. An interesting example of those actions can be seen in figure 10.2, where a unit may alter its own and the target minimum- and maximum ranges based on the health of the target and its own minimum attack range.

Combined fitness towards asymmetry ($\gamma = 0.75, \lambda = 0.25$) Combining the two fitnesses with these weights produced similar results than the previous combination: even though the simulation-based fitness might return a fitness of zero if the game is not winnable for one player, would a weight of $\gamma > 0$ always lead to a high score when

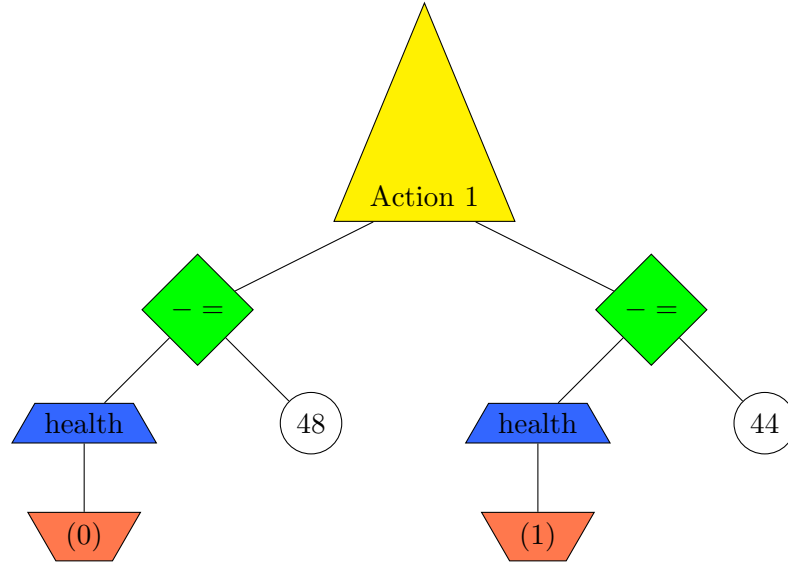


Figure 10.1: An action evolved solely through the Asymmetric fitness. Although it decreases a target's health, it also decreases its own healths in order to invoke the action. But doing so would kill the unit, as the ObjectClass only grants 44 health when a new unit spawns.

the two unit sets are very uneven. The observations made in the previous example do apply here as well. The highest scoring - and winnable - example, achieved an overall score of 0.39.

Equally weighted fitnesses ($\gamma = 0.5, \lambda = 0.5$) The highest scoring individual in this configuration (0.33) was winnable for both players. Even though the unequal distribution of available classes (one versus three) persisted, the simulation-base fitness with a weight of 0.5 already affected the population in a way that non-winnable individuals, i.e. games which achieved a score of zero through the balancing fitness, were decimated. Furthermore, the only class available to player one has a significant higher amount of health (90 versus 2, 2, and 28). In theory, player two has a variety of unit classes at his disposal, while player one has to rely on one strong class.

Combined fitness towards Balancing ($\gamma = 0.25, \lambda = 0.75$) This highest scoring individual (0.22) in this experiment assigned two classes to player one, and three classes to player two. All classes had a maximum health value between 30 and 70 hit points. The maximum ranges of player two's units were between 7 and 10, with a minimum range 2 points lower than the maximum range on average. One class for player

10. OVERALL RESULTS

one differed significantly: its minimum range was zero, making it the most versatile unit in the game in terms of range; but in return the class had the lowest health points of all unit types in the game (30 points) and could only heal (41 points per heal) other units. Player two had a healing unit as well at his disposal. This unit has a weaker healing ability (20 points), but could also heal itself significantly (39 points), and use a weak attack action for self-defence.

Only Balance ($\gamma = 0.0, \lambda = 1.0$) The high-scoring individuals in this experiment run were typically structured as follows: player one disposes of one unit type with a strong attack and the ability to heal itself quickly. Player two can utilise two unit types: one that can attack enemy units, stronger than the units of player one may, but posses no further abilities. But player two also has a unit type as his disposal that has no direct attack action but may boost other friendly units' minimum attack range (although this attribute is not used not as such) and heal other units. In summary, player one may only use units that can attack and heal themselves while player two has weaker units but - since every unit has one action per turn - may heal a unit that is attacking another target within the same turn. Ultimately, both unit sets have the same abilities, but with a little twist.

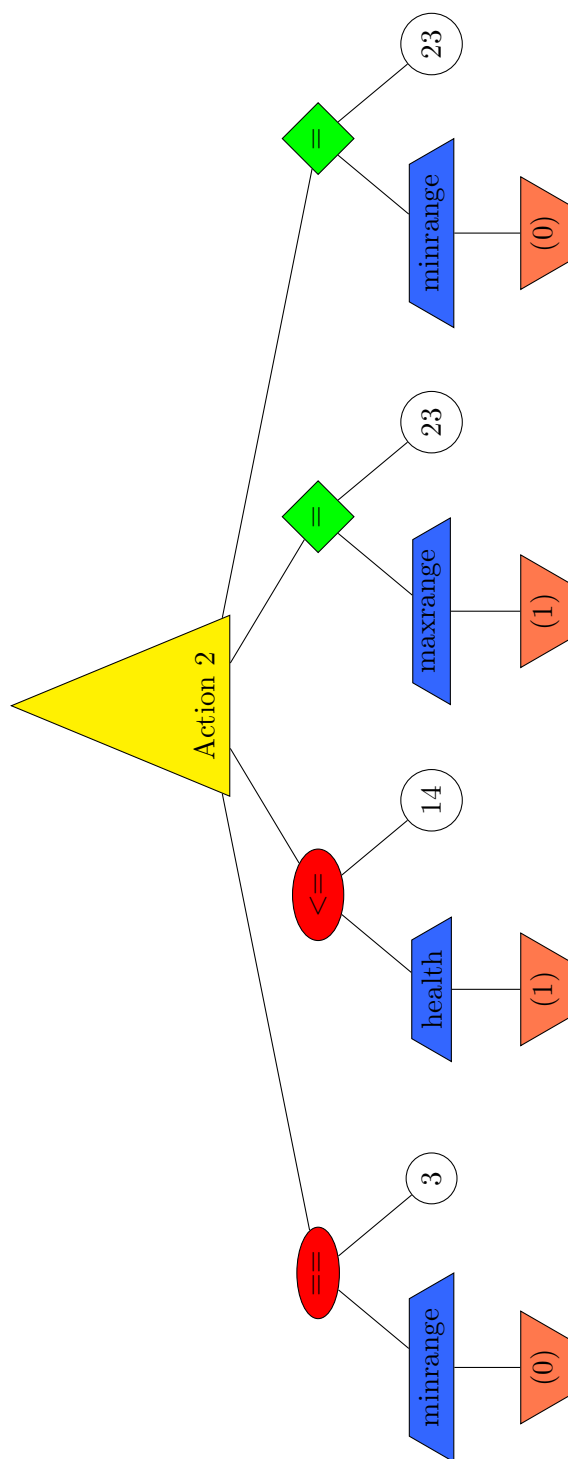


Figure 10.2: An Action which adjusts both the min- and maximum ranges of attacker and target

10. OVERALL RESULTS

| | Class A | Class B | Class C | Class D |
|-----------|---------|---------|---------|---------|
| Player | 1 | 2 | 2 | 2 |
| Speed | 2 | 3 | 1 | 2 |
| Min range | 5 | 8 | 0 | 0 |
| Max range | 13 | 10 | 6 | 5 |
| Health | 53 | 60 | 17 | 10 |
| Cost | 8 | 14 | 14 | 13 |

Table 10.2: Attributes of the classes found after 100 generations

10.1 Further experiments

Running exemplary experiments with different parameter weights was the first step in analysing the behaviour of the genetic algorithm. We repeated the last experiment (with $\gamma = 0.5, \lambda = 0.5$) to gain more statistical information. The progression of both the average fitness per generation and the average maximum fitness in five runs of the experiment can be seen in figure 10.4. It seems that the overall maximum converges around generation 30, but the overall population is continuously improved by presumably variations of the known best individuals. The error bars show the standard deviation between the five runs, resembling very similar progressions. But naturally the low size of repetitions has an effect on this. More conclusive statements would require more experiments, which are scheduled as future research.

Nevertheless, we wanted to analyse the fitness' longterm development in the experiment. But simply raising the maximum number of generations was not feasible, as the previously performed experiments already required several days of computing time each. To run a longer evolutionary run, we reduced the number of samples (games played) per individual from 200 down to 50 - accepting that this would increase the fitness noise - but in return increased the population size to 30. Using the same parameters ($\gamma = 0.5, \lambda = 0.5$), we conducted a final experiment over 100 generations. The fitness development can be seen in figure 10.3. It seems that the maximum fitness stagnates around generation 50. The best individual returned by the algorithm is listed in table 10.2 and will be analysed in the following.

We can observe that diversity between the two armies is maintained by giving player 2 access to three different (but also expensive) unit types, while player 1 only has access to one type. Player one relies on the cheapest unit in game which is also the unit type with the largest maximum range. Its minimum range, speed are average, and it has the second highest health value in the game. Overall, this unit type is a versatile

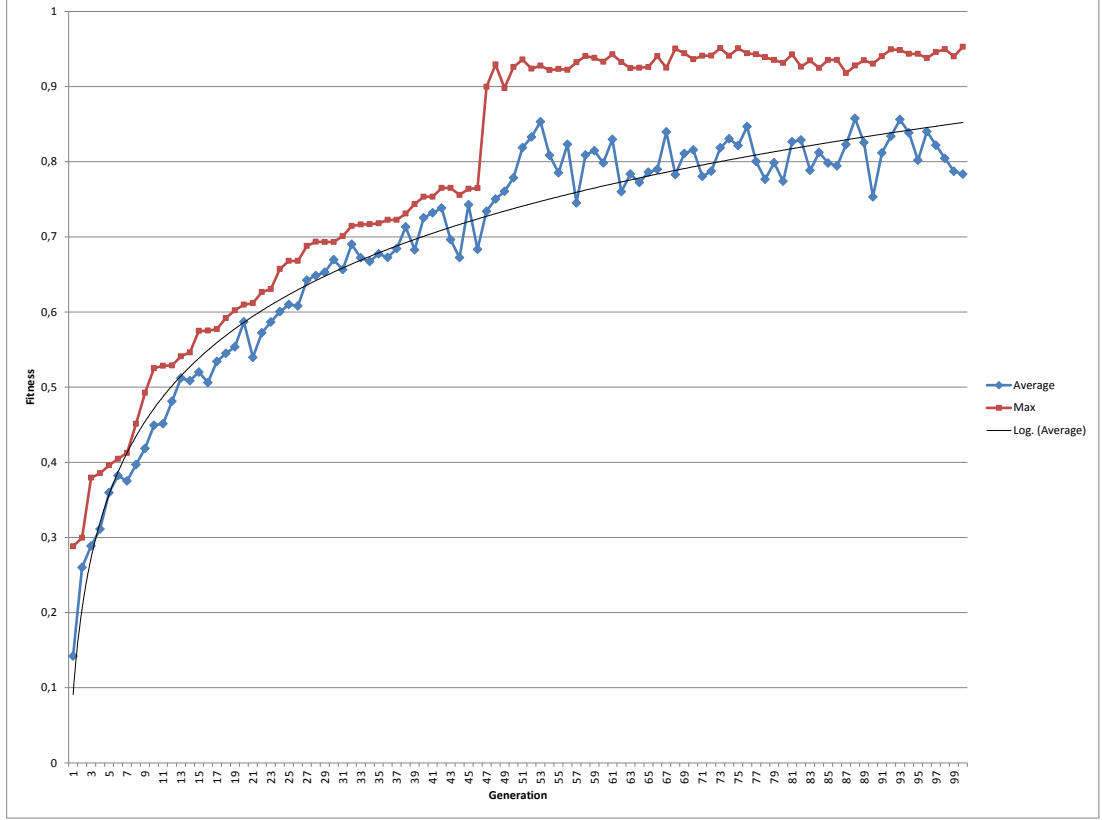
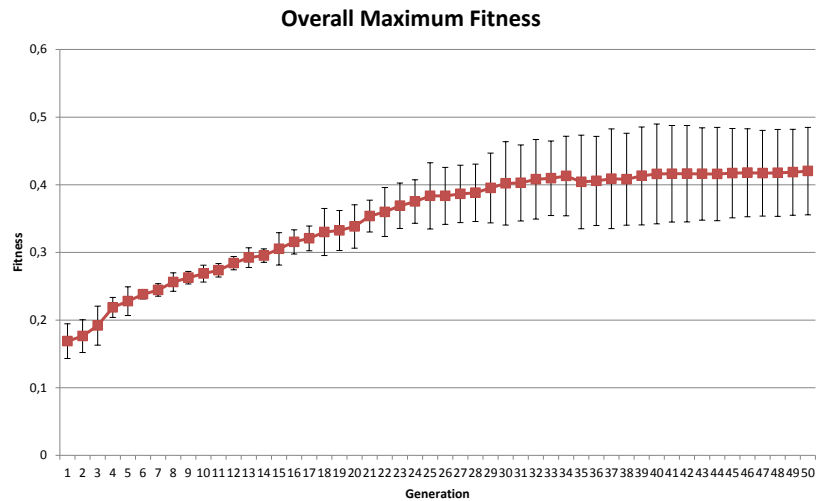


Figure 10.3: Fitness development over 100 generations with ($\gamma = 0.5, \lambda = 0.5$). The fitnesses of the first and last generation have a p-value $< .00000000001$

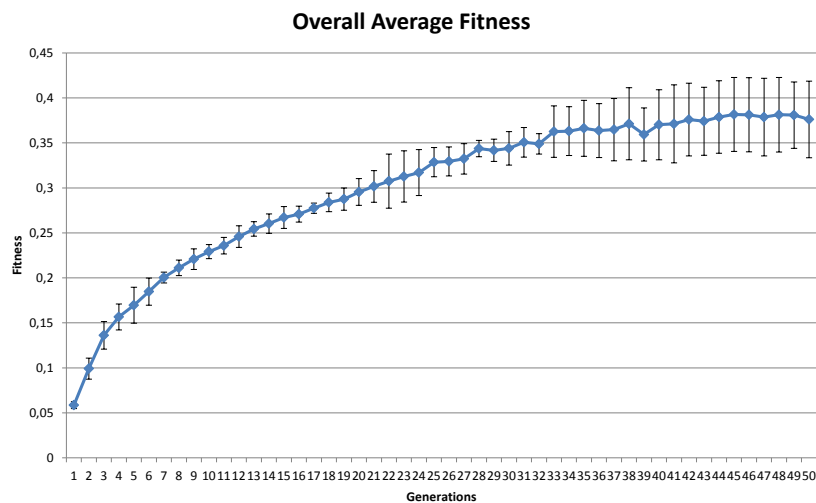
strong long range combat unit with a simple attack action using the constant value of 7. Furthermore, this unity type has an action to set its minimum range to 33 while lowering its maximum range by 44 - which seems quite ineffective. Player 2 may choose between the strongest (health) unit type in the game. This type is also the fastest in the game but has a very narrow min-/max-range window. It may also health itself by 20 points, and has a strong attack action with a strength of 21. However, it may only use said action if the target's health is below 21 and its own minimum range is not 42. It may increase its minimum range by 27 if its health is above 23. Although the values certainly require some manual tweaking, this is fairly the most interesting mechanic we have observed so far in the experiments.

The second unit type resembles a close combat unit with no minimum range, but its health value is below average. It does however possess a strong healing action which might be applied to other units. Interestingly, the target's maximum range must not exceed 27, and its health may not be 42. This unit type has also a minimal

10. OVERALL RESULTS



(a) Overall Maximum Fitness



(b) Overall average Fitness

Figure 10.4: Fitness developments over all five runs of the experiment using $\gamma = 0.5$, $\lambda = 0.5$. The overall maximum fitness depicts the average maximum fitness in each generation, the overall average fitness shows the average average fitness over all runs. The error bars represent the standard deviation in each generation.

attack action, restricted by several conditions on its own or the target's minimum- and maximum range.

Similar, the last unit type player 2 may chose from is slightly faster but has less health. The latter has an action which might be called "vampiric bite" it damages an opponent while healing itself. This action has also some restrictions on the unit and its target minimum range which seem rather uninteresting.

Overall, the generated actions seem very interesting but certainly require further optimisation of their values and conditions. A final comment should be made on the lowest individuals (< 0.4). Commonly these individuals can not be considered playable games. They are usually totally imbalanced, e.g. an individuals with the score of 0.012 had no attack actions available on the side of player one. Units on one player's side could just move around, while the other player was actually able to attack and win the game. The individual was only permitted for sampling as our model checks only searches for at least one attack action in the model. Although an inefficient allocation of computation resources, a low score was assigned by the balancing fitness; making it unlikely that this individual would be considered for procreation. We do not know what the minimum fitness in this case is to actually create playable games. It seems more efficient to change the model checks instead in way that they search for attack actions in both unit compositions. The individual was not assigned exactly zero, as some games definitely ended in a draw (due to the agent's inability to finish the game in time) and therefore the winrate of the dominating player is not exactly 100%.

10.2 Game in focus: a qualitative analysis

We would like to conclude the result section with a qualitative analysis of the playing experience provided by the generated games. Unfortunately, the human-experiment discussed in section 7.4 yielded no interesting or significant results regarding the game models. Furthermore, another user test was beyond the time frame of this dissertation. Instead we describe an exemplary process of taking a generated game model and playing it against our MCTS agent.

A first attempt to play a model emerges as impossible. Without meaningful annotations or descriptions available, the evolutionary algorithm assigns random names to unit classes and actions. A human player may discover the effects of each action by simply using the trial-and-error principle – yet, this is completely impractical. Instead, an inspection of the actual model is necessary – comparable with reading the manual of a game. By analysing the conditions and consequences of each action manually, we

10. OVERALL RESULTS

were able to annotate each action and class in a meaningful way.

It became evident that the whole game evolves around manipulating maximum ranges of the own units and the enemy units. At this point the game is playable for a human, but ironically – as no attack action includes a range condition – maximum ranges are meaningless, i.e. attack actions are global. The game becomes a puzzle with the same characteristics as we observed in our balancing experiment (204), i.e. the game is reduced to a puzzle of which units to build first and which action to use in the correct order.

Our next step was therefore the addition of range and ammunition condition to all actions. This creates the necessity for the combat units to approach the enemy. In a second step we divided all maximum and minimum ranges by four to scale them correctly to the map size.

The resulting game is an “arms race” in which the computer (or second) player tries to disable the player’s units by increasing their minimum range. In our test game, player one’s units have no ability to control their minimum range, therefore the units become stranded. Their advantage however is, that they are cheaper to produce than player two’s unit and therefore may outnumber those. This is also possible due to resetting the health of a single unit with a low maximum range that draws the enemy fire.

We have played a small number of games using this modified model, and use Ritchie’s criteria *novelty*, *quality*, and *typicality* to discuss the output.

Novelty We see the game as P-Creative (see section 4.2) to the generating software, as the population was not initialised with any kind of existing game models. The game can also be considered partly H-Creative, as - to our knowledge - no game that centres exactly around modifying shooting ranges like our game exists. But naturally, the aspect of disabling enemy weapons has been used in previous games.

Quality If we consider the raw output of the generator, no human player would enjoy the game due to the lack of intuitive gameplay. However, the manually refined version was reported as “acceptable” by a few test players.

Typicality The resulting gameplay is more on the “puzzle side” of strategy games, and seems typical for turn-based strategy games of the 1990s, e.g. *Battle Isle*, *Panzer General*, or *History Line: 1914-1918*. Modern games such as *Civilization V* rely more on secondary economic aspects, but other game series such as *Heroes of Might & Magic* retain their puzzle character in tactical combats.

10.3 Summary

Over the last two chapters we described how we combined the SGDL framework, including its automated gameplay capabilities, and the described fitness functions into a series of experiments to demonstrate the evolution of new strategy game mechanics. We started with the evolution of a simple set of parameters which affected the balance and playability of a simple game, compared the results of our fitness functions regarding different game models, and ultimately evolved new Actions - forming new unit types. The created mechanics are creative in a way, that our example models did not contain mechanics such as altering enemy units' max- or minimum ranges. We also saw, that the evolved actions' characters changed based on the weights of the two fitnesses.

We have shown that SGDL is capable to express strategy games and their common game elements, and that it can be used to evolve these further, allowing different types of gameplay. Even though we focussed on the evolution of Action subtrees, we believe that this process can be applied to the complete game tree. Our artificial agents were able to play new games within there defined boundaries, i.e. games which could be assessed using our state evaluator function. We concluded the presentation of the results by analysing a few game models and the resulting gamplay.

We will conclude with a discussion of the gained results and insights in the last chapter.

10. OVERALL RESULTS

Chapter 11

Beyond tactics: Conclusions and Discussion

Over the course of the last chapters we presented a method to specify the game mechanics of strategy games. As a prerequisite we tried to specify how strategy games can be distinguished from other games, and how they are seen in other literature, to form a definition of the genre which works for us. We then presented several experiments in which our framework was used to evolve certain aspects of strategy games' mechanics. The results indicate that our method can be used to generate interesting (strategy) games. This also implies, that the various definitions of “interesting” (fitness functions) used from other genres can be applied to strategy games. This seems natural given the similarities between board- or combinatorial games. However, several directions of future research lie ahead. The last chapter will therefore discuss the insights gained from our experiments, their limitations, and future ways of exploring this area of research.

11.1 Modelled Games

The games we modelled in SGDL so far can be described as “very basic” strategy games. They contain the basic elements which can be found in many strategy games despite lacking some common features such as hidden information. We focussed on the primary strategy game mechanics, the gameplay of warfare. And even though commercial games offer a larger variety of primary mechanics, e.g. more units types, different attacks, etc., we consider the modelled games as comparable to said games. As we modelled the basic interactions between units (attacking, moving, blocking, healing etc.) it would be easy

11. BEYOND TACTICS: CONCLUSIONS AND DISCUSSION

to add more unit types to our games. Although agents such as the XCS would require anew training, agents which rely on the state evaluator function could play games with additional unit types without a change. Our fitness function however would require computation time. Especially the “balancing” fitness’ complexity grows exponentially with new unit types.

As mentioned, the secondary mechanics (economic aspects; see chapter 2) are not as complex as in commercial games. Titles such as *Civilization* offer a complex model of economics and supplies. The games modelled so far therefore could be better described as “tactical games”, as the game mechanics focus on a micro level: moving units around and selecting the right target for shots than developing a good overall strategy. Logistics play a minor role in our games, what resembles in an interesting anecdote: a known flaw in *Rock Wars* (that players can enforce a draw by not building any units even with enough resources left) seemed to have no impact on the experiments since we did not encode a “skip turn” option for our agent. Although it may be beneficial in a strategy game to not act for a given time (e.g. to wait for more resources), we designed the agent to not break Huzinga’s magic circle, i.e. by refusing to play the agent would reject the game rules. But such a potential exploit would be critical for a multiplayer game. Finally, it could be easily fixed by making the factory destroyable.

We maintained the simple game mechanics mainly to allow an easier development of the general gameplaying agents. Even with low branching factors the runtime complexity, as the MCTS agent uses hundreds of playthroughs per turn. Agents which relied on learning techniques, e.g. the XCS agent, are much faster but require extensive training time.

However, we are confident that complexer games can be modelled in SGDL. The principle of the language allows modular addition of new features, especially node types.

11.2 New Game Mechanics

Even though our games focussed on tactical gameplay, we consider our generation experiments a success: we were able to generate new game mechanics for our base game *Rock Wars*. The algorithm was able to create game mechanics which were not part of the original game. The algorithm generated healer units, or units which altered other units’ minimum or maximum shooting ranges. If we set this into context of existing games, it is unknown if we actually created game mechanics. To speak in terms of computational creativity: our game mechanics can be perceived as creative, as they were unknown to the generator before. However, healer units or units which boni

or mali to other units have been seen in other games before. Our new game mechanics are there not historically creative. A more thorough investigation of the generated game mechanics and what kind of gameplay they allow seems in order.

The models we obtained, especially from the final experiments, need further manual tweaking and playtesting to form games which are playable and enjoyable by humans. But the algorithm was able to generate a basic game mechanics idea of using the minimum and maximum ranges as a condition for healing, which in return could only be altered by a second unit. If a human game designer recognises this pattern, we believe it could be formed into an actual playable game. This supports the argument, that SGDL can be used as an augmentation of the human creativity. The main direction of future research lies clearly in the validation of the enjoyability of the generated models, the question if the system generates games which can eventually be further processed by human game designers (and under what circumstances), and the possibility of including player preferences in the generation process. So far there is no adaptivity in our system, generating models only for a “default” user. The next intermediate step however would be the evolution of a whole SGDL tree, including object classes, attributes, and winning conditions - not just actions. To summarise, the generated models were interesting but unrefined. They could be played by artificial agents. They remain to be validated against human player preferences.

11.3 The Strategy Games Description Language

We were able to model large parts of a strategy game, started with evolving simple parameters, and were finally able to evolve subtrees of actions. However, this is only an intermediate step in evolving complete strategy games’ mechanics. As outlined above, our games and experiments focussed on the aspect of warfare. Furthermore, our experiments did not alter the unit classes’ number of attribute. Also the winning conditions remained unchanged, as this would have required a new state evaluator function. In our last experiments all relevant aspects were modelled in SGDL (as opposed to be programmed in Java code). Those behaviours which remained hardcoded in the game engine were solely the number of players, starting positions, turn order etc. As this is descriptive information, we believe that these features could be specified in the SGDL root node as Meta information as well.

Other things which were ignored so far are the aspect of partial information (“fog of war”) and non-deterministic elements. We were able to model games which followed our definition presented in chapter 2, and we believe that the language can be extended

11. BEYOND TACTICS: CONCLUSIONS AND DISCUSSION

sufficiently to capture the missing aspects of strategy games. The current shortcomings are more due to practical reasons and the project’s timescale rather than conceptual problems. For example, nodes which couldn’t be finalised at the time of the experiments which concerned non-deterministic effects, and area effects (“splash damage”) were already in the testing phase. Our point here is that the language is flexible enough to be extended on a case basis.

11.3.1 Verbosity vs. Versatility

Compared to the approaches presented in section 4.4, SGDL is very verbose, i.e. even simple concepts such as “move n tiles to the left” require a multitude of nodes. This is due to our design choice to model logic on a very low level. Thus even the SGDL trees of simple games like CPRS or Rock Wars tend to grow large. The SGDL tree for Rock Wars for example includes several hundred nodes; games like Civilization would probably require millions of nodes. This implies that human game designers can use SGDL only with the help of (visual) editing software. In practice, this is irrelevant once a working toolchain has been developed.

From our own experience, SGDL trees are easily comprehensible on a micro level, as described in the introduction of chapter 5. A person familiar with SGDL was able to quickly tell what an Action “does” just by looking at the subtree’s visual representation. On the other hand, we think that the highlevel design of SGDL trees requires additional discussion. Our internal tests showed, that even persons familiar with SGDL had difficulties to keep track of the functions of subtrees in larger models, and how unit types and other aspects interacted: our current visualiser for SGDL trees allows zooming in and out of SGDL trees. While it is easy to recognise the pattern of an “attack” action when all the required nodes are clearly visible on screen, it is harder when zoomed out. We discussed different approaches to make SGDL trees more accessible on macro level. One direction would be, to encode high-level concepts into their own node types, e.g. attack actions, and therefore substitute subtrees with a single node. This would give designers a “shortcut” to easily add basic – reoccurring – concepts into their game. And by reducing the numbers per SGDL model we would also reduce the number of ill-defined trees produced by our various evolutionary algorithms. The amount of potential meaningless subtrees, often referred to as “bloat”, affect the algorithm’s efficiency to drop very abstract or unplayable games. Ultimately, adding additional node types would not limit SGDL’s versatility, as behaviour – which is not encoded in a high-level node may still be encoded manually.

The conclusion here seems to be, that the increased verbosity as a result of the

desired versatility is merely a problem of representation. A different approach would be introducing high-level nodes only in the editor, so designers can group nodes on a visual level, while the actual tree still contains all the low-level nodes. This however would not solve the resource overhead of a model-based game engine, i.e. implementing the game mechanics directly in code would always be faster. This however would not be a problem for the everyday use of SGDL: prototyping and debugging of games. In early design phases of a game project, runtime performance may be negligible.

11.4 The SGDL Framework

The current version of the SGDL game engine has some shortcomings which prevent it from being released to the public in its current form. The main reasons are usability issues. Even though our game engine can load any game which is modelled in the current SGDL version, we have no intuitive mechanism to communicate what the abilities of units are to the player, or what his legal moves are. In our online experiment we tried to solve this by supplying a textual description of the game and a short tutorial to describe the game. Legal actions were communicated through context menus: whenever a player clicked on one of his units, the engine did a combinatorial test of all actions of that unit's class and all objects on the board. For each action which condition tree returned "true" an item was displayed in the context menu. This approach of using context menus breaks with the common genre convention that units, targets, etc. are directly selected with the mouse. Several players in our online experiment mentioned this aspect negatively. We also tested a mouse based approach for our Dune II prototype (which was not finalised at the time of writing). But the development of our Dune II prototype was halted, as the development of an flexible and usable interface from scratch consumed too much time in the project plan. It should be added though, that the context menu based approach enabled the player to use arbitrary actions modelled in SGDL – just not in an intuitive way. This was also caused by the fact, that the SGDL engine is not able to visualise effects of actions beyond textual log files ("Unit A's health is now 14"). This may be sufficient for debugging and prototyping a game, but it seems inappropriate for playtesting it. The problem here is, that the SGDL engine has no a priori knowledge about what an unit or action "does" to visualise it. We approached this problem by manually annotating the SGDL model used for our online experiment, specifying that attack actions should be visualised in the form of a laser beam. The SGDL engine is developed to a point, that a human game designer may manually specify which graphics (3D models, effects, etc.) should be used for unit- or building types or the effect of

11. BEYOND TACTICS: CONCLUSIONS AND DISCUSSION

actions. The automatic generation of asset sets and graphical user interfaces based on SGDL models would be an interesting direction of future research though.

11.5 Computational Complexity

Developing agents which can play the modelled games on an adequate level and simulate different playing styles posed a greater challenge than initially expected. But ultimately, we were able to develop agents which were able to play our agents on a sufficient level. The data gathered through our online experiment shows, that the agents were at least able to win 21% of games played and it is not trivial to beat them.

11.5.1 Status of the General Gameplaying Agents

The agents we developed are capable of playing the game and winning it. We saw different learning rates and performance in terms of win rate against bots and human players. All bots were capable of beating the heuristic and a random player significantly. The main limitations of most bots however is, that they rely on a board evaluator function that is specifically tailored to a certain objective: removing enemy units from the board, and winning the game by disposing all enemy units. Although board games exists where the players' goal is to lose all their game pieces, e.g. the chess variant *Antichess*¹. We consider these as the exceptions, and the authors are not aware of any commercial digital strategy game of this type. However, this remains a restriction.

The performance of our agents is limited by the high branching factor of a strategy game game tree. The large number of units combined with a large number of possible options make a complete search infeasible, and the main effort in this area of research should lie on the efficient pruning of unpromising move options. The claim that, "strategy" implies "considering all options" brought up in section 2, seems to fall short of practicality. While employing pruning techniques such as UCT help to reduce the computational costs, a full exploration of the possibility space is still impractical.

Human-likeness Another thing which was not fully explored during the development of the agent was the goal to create agents which imitate human play. While technically artificial agents mimic how human players interact with a game, our agents have no models of emotions or other physiological characteristics which could for example affect response times or overview of the game, i.e. our agents never forget to move a unit simply because it is not currently visible on screen. The goal to implement agents

¹The goal of the game is loose all game pieces but the king

which imitate human-like responses was not explored for practical reasons: balancing the workload between project members. It would be interesting though to create a cognitive and emotional model which can be applied to strategy games. The application of existing models (as presented in chapter 3) might be an option. Although we have seen some differentiation in how human players perceive bots as “human-like” (please refer to figure 7.7), the data sample of 60 players seems to small to draw generalisable conclusions. Yet, we can only speculate what caused these differences, i.e. how can a playing style be characterised as human-like, and how can it be consciously achieved. For example, the NEAT agent played a very defensive strategy in *Rock Wars* with simply lining up all his units in his base and waiting for the enemy. From personal experience of the authors, this is a common tactic used by novice players in strategy games, who like to dig themselves in and try to wear of the opponent. Although this is a tactic played by human players, the NEAT agent received a low “human-likeness” score (27.78%). As a conclusion, we see great potential for future research in human-like strategy game bots. It should be noted, that academic conferences¹² already organise bot competitions for various games. While the Starcraft competitions focus more on win-ability (at the time of writing all published bots could be beaten easily by expert players), the *2K Bot Prize*³ uses a setup for *Unreal Tournament* similar to a Turing-test. Similar to the original Turing-test, where participants have to guess if they are chatting with a human or if an algorithm is generating chat responses, players have to judge if the movement and actions in a first-person shooter are controlled by a bot or a human player.

11.6 Human studies

Another result of our study using human play-testers was that the enjoyability of the agent strongly correlated with the enjoyability of the selected SGDL model. The results of the question: “Disregarding the opponent, which game did you prefer?” were therefore removed from the analysis. We believe this was due to a flaw in the experiment setup: participants were not aware that the parameters of the game models were actually changing between games, and focussed solely on the agents’ behaviour. The differences between game models were too subtle. More conclusive results may be achieved by using SGDL models which significantly differ more from each other. The other possible conclusion, that the enjoyment purely depends on the opponent rather

¹CIG 2012 <http://ls11-www.cs.uni-dortmund.de/rts-competition/starcraft-cig2012>

²AIIDE 2012 <http://webdocs.cs.ualberta.ca/~cdavid/starcraftaicom/>

³2K Bot Prize <http://botprize.org/>

11. BEYOND TACTICS: CONCLUSIONS AND DISCUSSION

than the game played, seems unlikely. In practice, people will eventually develop a preference for one game over another even if they play against the same person.

The most frequent comment given by players in the free text comment field concerned the user interface (as discussed in section 11.4). It seems evident, that the shortcomings affected the playing experience in a negative way. Even though most players gave it the benefit of being an academic experiment rather than a commercial released and polished game. Most comments concerned the unintuitive action selection, and a context sensitive click system is considered a genre standard. As a conclusion, the next pending experiment, using our own implementation of *Dune II*, will therefore address these two main concerns: the lack of an intuitive interface, by implementing a variant of the original interface, and presenting participants two game models: a SGDL model of the original game, and a second one with a significant variation of the first model. This will enable us to validate the applicability of the fitness functions (presented in section 8) to strategy games. Our hypothesis is, that the fitness scores generated by our fitness functions will correlate with human players' preferences. Choosing only one agent for both games should remove the effect of the different playing styles.

11.7 Data Driven Evolution

The fitness functions used so far all rely on expert knowledge. Our own definition of "Balance" was based on our own experience and anecdotal evidence gained from strategy games' communities and media. Like the "thrill" measurement it has yet to be verified that the functions' results correlate with human players' preferences. Browne's "lead changes" fitness was originally tested against the preferences of human players, but has yet to be validated for strategy games. The flaw in our experiment with human players, as described in the previous section, prevented us from gaining significant insights on these questions. Expert knowledge may be biased and may not resemble other players' preferences and skill levels. Some players prefer an easily beatable opponent, other players demand a challenge, some players enjoy a slow pace game where they can focus on building an aesthetic pleasing base, and other players focus on fast paced tactical combat. The list of possible dimensions could be extended in various ways. Our point here is, that it may be beneficial to connect the research on strategy game mechanics generation with the field of player experience modelling. Models of emotions or motivations may help to generate strategy games which are tailored to a specific player or just more appealing to players in general. Possible data sources found in published research range from self-reported data, physiological

measurements (94, 206), gaze data (207) and qualitative case-studies (208, 209) to automatically collected gameplay data from within the game. The application of machine learning techniques to in-game data (also: game metrics mining) has become popular in recent times (210, 211, 212). Data mining algorithms have been proven to be successful in eliciting new information about how players interact with the game world. As we have published in this area of research ourselves (213), we believe that this would be a promising direction of future research; to validate the fitness functions used in our previous research in this area.

We envision an extended system, based upon the knowledge we gathered in our initial experiments: a development kit based on SGDL which enables interested individuals to create and share strategy games on a dedicated website. Interested players can use our game engine to download and play the shared games. As the platform will be open to the public the roles of *designers* and *players* interchangeable, forming a gaming community. Additionally, the shared games can be used as a basis for our evolutionary process.

There are two additional features we envision: an integrated instrumentation framework which provides detailed quantitative feedback about the playing experience. The gathered player metrics will be analysed with state of the art computational intelligence/data mining techniques. And secondly, the created games can be used as a competition framework for artificial agents. Similar to the Starcraft competitions mentioned earlier in this chapter, we believe that creating agents for arbitrary strategy games might be an interesting challenge for programmers.

11.8 Lessons learned

The work presented in this thesis demonstrates that it is generally possible to generate strategy game mechanics using an evolutionary algorithm. The domain specific language (SGDL) is capable of representing games on the required level of detail. The agent framework is able to play a set of games (the ones where our board evaluator function is applicable to) but still can't handle "any" game expressed in SGDL.

The computational costs to learn or even to play a game are enormous. Especially the experiments with the "thrilling" fitness consumed massive amounts of computation time. The computational cost has been underestimated at some points in the project.

The evolutionary experiments showed, that the SGDL trees suffer from the known problem of "bloat", the fill of SGDL trees with irrelevant nodes. These models initially wasted a lot of computing time until we implemented the offline model checks.

11. BEYOND TACTICS: CONCLUSIONS AND DISCUSSION

Overall, fitness functions that take such long time to compute make it difficult to experiment with the parameters of the evolutionary algorithm. Investing more time in developing offline model checks would make the evolutionary process more efficient, but implementing domain specific knowledge might also reduce the diversity of the result it produces.

The Dune 2 map generator is a nice addition to the project, but the time might have been better spent on the optimisation of the framework. Nevertheless, it demonstrates from a different angle, that commercial strategy games can be expressed in SGDL. Overall, we should consider that this is a research project and not product development. Some turns and twists in the milestone planning seem acceptable.

From a technical point of view, the decision to implement everything in Java was the right one; especially in regard of the project members knowledge of the technology. This made it also comparably easy to create the online experiment, as we just transferred the SGDL engine into an applet context. Ultimately, we do not recommend of implementing such a complex project in any unmanaged language, even though this is clearly possible.

11.9 Concluding Remarks

The SGDL framework presented in this thesis stands as initial research in the area of strategy games generation. The modelling capability and the results gained from our experiments are promising and suggest further research in this area. We believe, that the insights gained in our research should encourage researchers and industry alike to develop further into the area of personalised generated game content, game mechanics in particular. We demonstrated that it is possible to generate aspects of strategy games gameplay using expert knowledge, and argued that our findings could be transferred to commercially produced games as well. Yet, the used fitness functions require improved framing; game metrics mining from online player data seems one potential future direction. Given the success of commercial success of *Zillion of Games*, *Game Maker*, and other game creation kits - we believe a release of the SGDL tool-kit to an interested public would be large success. Afterall, strategy games are a popular genre, and no comparable specialised products seem to exist.

11. BEYOND TACTICS: CONCLUSIONS AND DISCUSSION

References

- [1] MARTIN LORBER. **Transformation der Industrie**, Aug 2012. Available from: <http://spielkultur.ea.de/kategorien/wirtschaft/transformation-der-industrie>. 2
- [2] GERMAN TRADE ASSOCIATION OF INTERACTIVE ENTERTAINMENT SOFTWARE (BIU). **Die deutsche Gamesbranche 2011**, 2012. Available from: <http://www.biu-online.de/de/fakten/marktzahlen/die-deutsche-gamesbranche-2011.html>. 2
- [3] JULIAN TOGELIUS, GEORGIOS N. YANNAKAKIS, KENNETH O. STANLEY, AND CAMERON BROWNE. **Search-Based Procedural Content Generation**. In *EvoApplications*, pages 141–150, 2010. 3, 5, 50
- [4] JENNIFER JOHNS. **Video games production networks: value capture, power relations and embeddedness**. *Journal of Economic Geography*, **6**:151–180, 2005. 3
- [5] STEVEN POOLE. **Bang, bang, you're dead: how Grand Theft Auto stole Hollywood's thunder**. Online, Mar 2012. Available from: <http://www.guardian.co.uk/technology/2012/mar/09/grand-theft-auto-bang-bang-youre-dead>. 3
- [6] L.A. TIMES. **Star Wars: The Old Republic — the story behind a galactic gamble**. Online, Jan 2012. Available from: <http://herocomplex.latimes.com/2012/01/20/star-wars-the-old-republic-the-story-behind-a-galactic-gamble/#/0>. 3
- [7] JASON WILSON. **Indie Rocks! Mapping Independent Video Game Design**. *Media International Australia, Incorporating Culture & Policy*, **115**:109–122, May 2005. 3
- [8] GEORGIOS N. YANNAKAKIS AND JULIAN TOGELIUS. **Experience-Driven Procedural Content Generation**. *Affective Computing, IEEE Transactions on*, **2**(3):147–161, July 2011. 5
- [9] B.G. WEBER, M. MATEAS, AND A. JHALA. **Using data mining to model player experience**. In *FDG Workshop on Evaluating Player Experience in Games*, Bordeaux, June 2011. ACM. 5
- [10] C. PEDERSEN, J. TOGELIUS, AND G.N. YANNAKAKIS. **Modeling Player Experience for Content Creation**. *IEEE Transactions on Computational Intelligence and AI in Games*, **2**(1):54–67, march 2010. 5
- [11] JOHAN HAGELBÄCK AND STEFAN JOHANSSON. **A Multi-agent Potential Field based bot for a Full RTS Game Scenario**. *International Journal of Computer Games Technology*, **2009**:1–10, 2009. 9, 120
- [12] G. SYNNAEVE AND P. BESSIERE. **A Bayesian model for opening prediction in RTS games with application to StarCraft**. In *Computational Intelligence and Games (CIG), 2011 IEEE Conference on*, pages 281–288, 31 2011-sept. 3 2011. 9
- [13] BEN G. WEBER, MICHAEL MATEAS, AND ARNAV JHALA. **Applying Goal-Driven Autonomy to StarCraft**. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2010. 9
- [14] M. VAN DER HEIJDEN, S. BAKKES, AND P. SPRONCK. **Dynamic formations in real-time strategy games**. In *IEEE Symposium On Computational Intelligence and Games*, pages 47–54, December 2008. 9
- [15] ISTVÁN SZITA, GUILLAUME CHASLOT, AND PIETER SPRONCK. **Monte-Carlo Tree Search in Settlers of Catan**. In H. VAN DEN HERIK AND PIETER SPRONCK, editors, *Advances in Computer Games*, **6048** of *Lecture Notes in Computer Science*, pages 21–32. Springer Berlin / Heidelberg, 2010. 9, 57
- [16] ROBIN BAUMGARTEN, SIMON COLTON, AND MARK MORRIS. **Combining AI Methods for Learning Bots in a Real-Time Strategy Game**. *International Journal of Computer Games Technology*, **2009**:10, 2009. 9, 63
- [17] S. WENDER AND I WATSON. **Using reinforcement learning for city site selection in the turn-based strategy game Civilization IV**. In *IEEE Symposium Computational Intelligence and Games*, pages 372–377, 2008. 9
- [18] DAVID AHA, MATTHEW MOLINEAUX, AND MARC PONSEN. **Learning to Win: Case-Based Plan Selection in a Real-Time Strategy Game**. In HÉCTOR MUÑOZ-ÁVILA AND FRANCESCO RICCI, editors, *Case-Based Reasoning Research and Development*, **3620** of *Lecture Notes in Computer Science*, pages 5–20. Springer Berlin / Heidelberg, 2005. 9
- [19] D. CHURCHILL AND M. BURO. **Build Order Optimization in StarCraft**. In *Seventh Artificial Intelligence and Interactive Digital Entertainment Conference*, 2011. 9
- [20] ALBERTO URIARTE PÉREZ. **Multi-Reactive Planning for Real-Time Strategy Games**. Master's thesis, Universitat Autònoma de Barcelona, 2011. 9
- [21] PEDRO CADENA AND LEONARDO GARRIDO. **Fuzzy Case-Based Reasoning for Managing Strategic and Tactical Reasoning in StarCraft**. In ILDIR BATYRSHIN AND GRIGORI SIDOROV, editors, *Advances in Artificial Intelligence*, **7094** of *Lecture Notes in Computer Science*, pages 113–124. Springer Berlin / Heidelberg, 2011. 9
- [22] MARK CLAYPOOL. **The effect of latency on user performance in Real-Time Strategygames**. *Computer Networks*, **49**(1):52–70, September 2005. 9

REFERENCES

- [23] CHRIS CHAMBERS, WU-CHANG FENG, WU-CHI FENG, AND DEBANJAN SAHA. **Mitigating information exposure to cheaters in real-time strategy games.** In *Proceedings of the international workshop on Network and operating systems support for digital audio and video*, NOSS-DAV '05, pages 7–12, New York, NY, USA, 2005. ACM. 9
- [24] HAROLD J. MURRAY. *The History of Chess*. Oxford University Press, 1913. 10
- [25] JOHN M. ROBERTS, MALCOLM J. ARTH, AND ROBERT R. BUSH. **Games in Culture.** *American Anthropologist*, **61**(4):597–605, 1959. 10
- [26] J. HUIZINGA. *Homo Ludens: Proeve eener bepaling van het spel-element der Cultuur*. Amsterdam University Press, 2010. 10, 134
- [27] ROLF F. NOHR AND SERJOSCHA WIEMER. **Strategie Spielen.** In *Strategie Spielen*, Braunschweiger Schriften zur Medienkultur, pages 7–27. Lit Verlag Berlin, 2008. 10, 149
- [28] NIKLAS LUHMANN. **Die Praxis der Theorie.** *Soziale Welt*, **20**(2):129–144, 1969. 10
- [29] NORBERT ELIAS. *Über den Prozess der Zivilisation : soziogenetische und psychogenetische Untersuchungen / Norbert Elias*. Haus Zum Palken, Basel :, 1939. 11
- [30] ANTONIO DAMASIO. *Descartes' Error: Emotion, Reason, and the Human Brain*. Harper Perennial, 1995. Available from: <http://www.worldcat.org/isbn/014303622X>. 11
- [31] OSKAR MORGENSTERN AND JOHN VON NEUMANN. *Theory of games and economic behavior*. Princeton University Press, 1944. 11
- [32] ROLF F. NOHR. **Krieg auf dem Fussboden.** In *Strategie Spielen*, Braunschweiger Schriften zur Medienkultur, pages 29–68. Lit Verlag Berlin, 2008. 12, 14, 16
- [33] THOMAS LEMKE. **Foucault, Governmentality, and Critique.** *Rethinking Marxism*, **14**(3):49–64, 2002. 12
- [34] J. LINK. *Versuch über den Normalismus: Wie Normalität produziert wird*. Historische Diskursanalyse der Literatur. Vandenhoeck & Ruprecht, 2009. 13
- [35] RAMÓN REICHERT. **Goverment-Games und Gouvernementainment.** In *Strategie Spielen*, Braunschweiger Schriften zur Medienkultur, pages 189–212. Lit Verlag Berlin, 2008. 13
- [36] MARCUS POWER. **Digitized virtuosity : video war games and post-9/11 cyber-deterrence.** *Security dialogue*, **38**(2):271–288, June 2007. 13
- [37] SEBASTIAN DETERDING. **Wohnzimmerkriege.** In *Strategie Spielen*, pages 29–68. Lit Verlag Berlin, 2008. 13, 14, 15, 17
- [38] J.C.L. HELLWIG. *Versuch eines aufs schachspiel gebaueten taktischen spiels von zwey und mehrern personen zu spielen*. S. L. Crusius, 1780. Available from: <http://books.google.dk/books?id=Gi8XAAAAAYAAJ>. 14
- [39] O. BÜSCH AND W. NEUGEBAUER. *Moderne Preussische Geschichte 1648-1947*. de Gruyter, 1981. 14
- [40] PETER H. WILSON. **Social Militarization in Eighteenth-Century Germany.** *German History*, **18**(1):1–39, 2000. 15
- [41] JAMES F. DUNNINGHAM. *Wargames Handbook*. iUniverse, 3rd edition, January 2000. 15
- [42] PETER PERLA. *The Art of Wargaming: A Guide for Professionals and Hobbyists*. US Naval Institute Press, 1st edition, March 1990. 15
- [43] ORIGINS GAME FAIR. **Origins 2012 Recap.** Online. [Online; accessed 17-September-2012]. Available from: <http://www.originsgamefair.com/>. 15
- [44] J. R. R. TOLKIEN. *The lord of the rings*. Allen & Unwin, London, 2nd edition, 1966. 16
- [45] MARSHALL MCLUHAN. *Die magischen Kanäle*. Verlag der Kunst Dresden, 2nd edition edition, 1995. 17
- [46] CHRIS CRAWFORD. **The Future of Wargaming.** *Computer Gaming Worlds*, **1**(1):3–7, 1981. 18
- [47] JON LAU NIELSEN, BENJAMIN FEDDER JENSEN, TOBIAS MAHLMANN, JULIAN TOGELIUS, AND GEORGIOS N. YANNAKAKIS. *AI for General Strategy Game Playing*. IEEE Handbook of Digital Games. IEEE, 2012. 18, 109, 149
- [48] NICOLE LAZZARO. *The Four Fun Keys*, pages 317–343. Morgan Kaufmann Pub, 2008. 24
- [49] THOMAS W. MALONE. **What makes things fun to learn? heuristics for designing instructional computer games.** In *Proceedings of the 3rd ACM SIGSMALL symposium and the first SIGPC symposium on Small systems*, SIGSMALL '80, pages 162–169, New York, NY, USA, 1980. ACM. 24
- [50] J. JUUL. *Half-real: Video games between real rules and fictional worlds*. The MIT Press, 2005. 25
- [51] WIKIPEDIA. **Snakes and Ladders — Wikipedia, The Free Encyclopedia**, 2011. [Online; accessed 15-September-2011]. Available from: http://en.wikipedia.org/w/index.php?title=Snakes_and_Ladders. 25
- [52] C.E. SHANNON AND W. WEAVER. *The mathematical theory of information*. University of Illinois Press, 1949. 25, 37, 142
- [53] WJNAND ISSSELSTEIJN, YVONNE DE KORT, KAROLIEN POELS, AUDRIUS JURGELIONIS, AND FRANCESCO BELLOTTI. **Characterising and Measuring User Experiences in Digital Games.** In *International Conference on Advances in Computer Entertainment*, June 2007. 25
- [54] WIKIPEDIA. **SpaceChem — Wikipedia, The Free Encyclopedia**, 2011. [Online; accessed 15-August-2011]. Available from: <http://en.wikipedia.org/w/index.php?title=SpaceChem>. 25
- [55] RICHARD BARTLE. **Hearts, Clubs, Diamonds, Spades: Players Who Suit MUDs**, 1996. 25
- [56] MATTHIAS RAUTERBERG. **Enjoyment and Entertainment in East and West.** In MATTHIAS RAUTERBERG, editor, *Entertainment Computing – ICEC 2004*, **3166** of *Lecture Notes in Computer Science*, pages 176–181. Springer Berlin / Heidelberg, 2004. 25

REFERENCES

- [57] WIJNAND IJSSELSTEIJN, WOUTER VAN DEN HOOGEN, CHRISTOPH KLIMMT, YVONNE DE KORT, CRAIG LINDLEY, KLAUS MATHIAK, KAROLIEN POELS, NIKLAS RAVAJA, MARKO TURPEINEN, AND PETER VORDERER. **Measuring the Experience of Digital Game Enjoyment**. In ANDREW SPINKMECHTELD BALLINTIJN NATASJA BOGERS FABRIZIO GRIECOLEANNE LOIJENS LUCAS NOLDUS GONNY SMIT PATRICK ZIMMERMAN, editor, *Measuring Behavior 2008*, International Conference on Methods and Techniques in Behavioral Research, Maas-tricht, The Netherlands, August 2008. 26
- [58] R. LIKERT. **A technique for the measurement of attitudes**. *Archives of Psychology*, **22**(140):1–55, 1932. 26
- [59] G. N. YANNAKAKIS. **Preference Learning for Affective Modeling**. In *Proceedings of the Int. Conf. on Affective Computing and Intelligent Interaction*, Amsterdam, The Netherlands, September 2009. 26
- [60] BASHAR NUSEIBEH AND STEVE EASTERBROOK. **Requirements engineering: a roadmap**. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 35–46, New York, NY, USA, 2000. ACM. 27
- [61] BETTY H. C. CHENG AND JOANNE M. ATLEE. **Research Directions in Requirements Engineering**. In *2007 Future of Software Engineering*, FOSE '07, pages 285–303, Washington, DC, USA, 2007. IEEE Computer Society. 27
- [62] BRONISLAW MALINOWSKI. *Argonauts Of The Western Pacific*. George Routledge And Sons, Limited, 1932. 27
- [63] C. GEERTZ. *Deep Play: Notes on the Balinese Cockfight*. American Academy of Arts and Sciences, 1972. 27
- [64] TIMOTHY C. LETHBRIDGE, SUSAN ELLIOTT SIM, AND JANICE SINGER. **Studying Software Engineers: Data Collection Techniques for Software Field Studies**. *Empirical Software Engineering*, **10**(3):311–341, 2005. 27
- [65] MARK CHEN. *Leet Noobs: Expertise and Collaboration in a "World of Warcraft" Player Group as Distributed Sociomaterial Practice*. PhD thesis, University of Washington, 2010. 28
- [66] TL TAYLOR AND E. WITKOWSKI. **This is how we play it: what a mega-LAN can teach us about games**. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, pages 195–202. ACM, 2010. 28
- [67] J. JUUL. *A Casual Revolution: Reinventing Video Games and Their Players*. MIT Press, 2010. 28
- [68] C.A. LINDLEY AND C. SENNERSTEN. **A cognitive framework for the analysis of game play: tasks, schemas and attention theory**. In *Workshop on the cognitive science of games and game play, the 28th annual conference of the cognitive science society*, 2006. 29
- [69] C.A. LINDLEY AND C. SENNERSTEN. **Game play schemas: from player analysis to adaptive game mechanics**. *International Journal of Computer Games Technology*, **2008**:1–7, 2008. 29
- [70] C.A. LINDLEY, L. NACKE, AND C.C. SENNERSTEN. **Dissecting play-investigating the cognitive and emotional motivations and affects of computer game-play**. In *CGAMES08*. Citeseer, 2008. 29
- [71] MIHALY CSIKSZENTMIHALYI. *Flow: The Psychology of Optimal Experience*. Harper Perennial, New York, NY, March 1991. 29
- [72] MARIOS KOUFARIS. **Applying the Technology Acceptance Model and Flow Theory to Online Consumer Behavior**. *Information Systems Research*, **13**(2):205–223, 2002. 30
- [73] JAWAID A. GHANI AND SATISH P. DESHPANDE. **Task Characteristics and the Experience of Optimal Flow in Human—Computer Interaction**. *The Journal of Psychology*, **128**(4):381–391, 1994. 30
- [74] PENELOPE SWEETSER AND PETA WYETH. **GameFlow: a model for evaluating player enjoyment in games**. *Comput. Entertain.*, 3:3–3, July 2005. 30, 33
- [75] PENELOPE SWEETSER, DANIEL JOHNSON, PETA WYETH, AND ANNE OZDOWSKA. **GameFlow heuristics for designing and evaluating real-time strategy games**. In *Proceedings of The 8th Australasian Conference on Interactive Entertainment: Playing the System*, IE '12, pages 1:1–1:10, ew York, NY, USA, 2012. ACM. 31
- [76] RAPH KOSTER. *A theory of fun for game design*. Paraglyph press, 2005. 31, 44, 45, 142
- [77] BARRY KORT, ROB REILLY, AND ROSALIND W. PICARD. **An Affective Model of Interplay Between Emotions and Learning: Reengineering Educational Pedagogy-Building a Learning Companion**. In *In*, pages 43–48. IEEE Computer Society, 2001. 31
- [78] HOPE R. CONTE AND ROBERT PLUTCHIK. **A circumplex model for interpersonal personality traits**. *Journal of Personality and Social Psychology*, **40**(4):701 – 711, 1981. 31
- [79] K. LEIDELMEIJER. *Emotions : an experimental approach / Kees Leidelmeijer*. Number 9. Tilburg University Press, [Tilburg] :, 1991. 31
- [80] PAUL EKMAN. **An argument for basic emotions**. *Cognition & Emotion*, **6**(3):169–200, 1992. 31
- [81] R W PICARD, S PAPERT, W BENDER, B BLUMBERG, C BREAZEL, D CAVALLO, T MACHOVER, M RESNICK, D ROY, AND C STROHECKER. **Affective Learning — A Manifesto**. *BT Technology Journal*, **22**:253–269, 2004. 31
- [82] ROSALIND W. PICARD AND JONATHAN KLEIN. **Computers that recognise and respond to user emotion: theoretical and practical implications**. *Interacting with Computers*, **14**(2):141 – 169, 2002. 31
- [83] M. MORI, K.F. MACDORMAN (TRANSLATOR), AND T. MINATO (TRANSLATOR). **The uncanny valley**. *Energy*, **7**(4):33–35, 2005. 32
- [84] SCOTT BRAVE, CLIFFORD NASS, AND KEVIN HUTCHINSON. **Computers that care: investigating the effects of orientation of emotion exhibited by an embodied computer agent**. *International Journal of Human-Computer Studies*, **62**(2):161 – 178, 2005. 32

REFERENCES

- [85] N. RAVAJA, M. SALMINEN, J. HOLOPAINEN, T. SAARI, J. LAARNI, AND A. J. "ARVINEN. **Emotional response patterns and sense of presence during video games: potential criterion variables for game design.** In *Proceedings of the third Nordic conference on Human-computer interaction*, pages 339–347. ACM, 2004. 32
- [86] LAURA ERMI AND FRANS MÄYRÄ. **Fundamental components of the gameplay experience: Analysing immersion.** *Proceedings of the DiGRA Conference*, **18**(35):15–27, 2005. 33
- [87] EMILY BROWN AND PAUL CAIRNS. **A grounded investigation of game immersion.** In *CHI '04 extended abstracts on Human factors in computing systems*, CHI EA '04, pages 1297–1300, New York, NY, USA, 2004. ACM. 33
- [88] GORDON CALLEJA. *Digital games as designed experience: Reframing the concept of immersion.* PhD thesis, Victoria University of Wellington, 2007. 33
- [89] G.N. YANNAKAKIS AND J. HALLAM. **Evolving opponents for interesting interactive computer games.** *From animals to animats*, **8**:499–508, 2004. 34
- [90] G.M. WILSON AND M.A. SASSE. **Investigating the impact of audio degradations on users: Subjective vs. Objective assessment methods.** In *Proceedings of OZCHI*, **4**, pages 135–142. Citeseer, 2000. 36
- [91] G.M. WILSON. **Psychophysiological indicators of the impact of media quality on users.** In *CHI'01 extended abstracts on Human factors in computing systems*, pages 95–96. ACM, 2001. 36
- [92] ANDERS DRACHEN, LENNART E. NACKE, GEORGIOS YANNAKAKIS, AND ANJA LEE PEDERSEN. **Correlation between heart rate, electrodermal activity and player experience in first-person shooter games.** In *Proceedings of the 5th ACM SIGGRAPH Symposium on Video Games*, Sandbox '10, pages 49–54, New York, NY, USA, 2010. ACM. 36
- [93] REGAN L. MANDRYK AND KORI M. INKPEN. **Physiological indicators for the evaluation of co-located collaborative play.** In *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, CSCW '04, pages 102–111, New York, NY, USA, 2004. ACM. 37
- [94] G.N. YANNAKAKIS AND J. HALLAM. **Entertainment modeling through physiology in physical play.** *International Journal of Human-Computer Studies*, **66**(10):741–755, 2008. 37, 183
- [95] CAMERON BROWNE. *Automatic generation and evaluation of recombination games.* PhD thesis, Queensland University of Technology, 2008. 38, 69, 144
- [96] GEORGE DAVID BIRKHOFF. *Aesthetic Measure.* Harvard University Press, 1933. 38
- [97] BARNEY DARRYL PELL. **Strategy Generation and Evaluation for Meta-Game Playing.** Technical report, Trinity College; in the University of CambridgeS, 1993. 38
- [98] WIKIPEDIA. **Dynamic game difficulty balancing — Wikipedia, The Free Encyclopedia**, 2011. [Online; accessed 16-September-2011]. Available from: http://en.wikipedia.org/w/index.php?title=Dynamic_game_difficulty_balancing. 40
- [99] RYAN B. HAYWARD AND JACK VAN RIJSWIJCK. **Hex and combinatorics.** *Discrete Mathematics*, **306**(19-20):2515 – 2528, 2006. 40
- [100] J. MARK THOMPSON. **Defining the Abstract.** The Games Journal, 2007. [Online; accessed 16-September-2011]. Available from: <http://www.thegamesjournal.com/articles/DefiningtheAbstract.shtml>. 41
- [101] JÜRGEN SCHMIDHUBER. **Formal Theory of Creativity, Fun, and Intrinsic Motivation (1990-2010).** *Autonomous Mental Development, IEEE Transactions on*, **2**(3):230 –247, sept. 2010. 42, 43
- [102] M. LI AND P. VITANYI. *An introduction to Kolmogorov complexity and its applications.* Springer, 1993. 42
- [103] JÜRGEN SCHMIDHUBER. **Developmental robotics, optimal artificial curiosity, creativity, music, and the fine arts.** *Connection Science*, **18**(2):173–187, 2006. 43
- [104] J. TOGELIUS AND J. SCHMIDHUBER. **An experiment in automatic game design.** In *Computational Intelligence and Games, 2008. CIG '08. IEEE Symposium On*, pages 111 –118, dec. 2008. 44, 71
- [105] JÜRGEN SCHMIDHUBER, JIEYU ZHAO, AND MARCO WIERING. **Simple Principles Of Metalearning.** Technical report, SEE, 1996. 44
- [106] S. R. K. BRANAVAN, DAVID SILVER, AND REGINA BARZILAY. **Learning to win by reading manuals in a Monte-Carlo framework.** In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1, HLT '11*, pages 268–277, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics. 44, 60
- [107] MIGUEL SICART. **Defining Game Mechanics.** *Game Studies*, **8**(2):1–14, 2008. Available from: <http://gamestudies.org/0802/articles/sicart>. 44, 47
- [108] ELLIOT M. AVEDON. *The Study of Games*, chapter The Structural Elements of Games. New York, NY: John Wiley & Sons, 1971. 45
- [109] A. JÄRVINEN. *Games without frontiers: methods for game studies and design.* VDM, Verlag Dr. Müller, 2009. 45, 47
- [110] KATIE SALEN AND ERIC ZIMMERMAN. *Rules of play: Game design fundamentals.* MIT Press, Boston, 2003. 45, 46, 47, 134
- [111] J. EDWARD RUSSO, KURT A. CARLSON, AND MARGARET G. MELOY. **Choosing an Inferior Alternative.** *Psychological Science*, **17**(10):899–904, 2006. 45
- [112] JOSÉ PABLO ZAGAL, MICHAEL MATEAS, CLARA FERNÁNDEZ-VARA, BRIAN HOCHHALTER, AND NOLAN LICHTI. **Towards an Ontological Language for Game Analysis.** In *DIGRA Conf.*, 2005. 45

REFERENCES

- [113] ESPEN ÅRSETH, LEV MANOVICH, FRANS MÄYRÄ, KATIE SALEN, AND MARK J. P. WOLF. "Define Real, Moron!" - Some Remarks on Game Ontologies. In *DI-GAREC Keynote-Lectures 2009/10*, 6, pages 50–68. Universitätsverlag Potsdam, 2011. 45
- [114] H.H. HOOS AND T. STÜTZLE. *Stochastic local search: Foundations and applications*. Morgan Kaufmann, 2005. 49
- [115] F. GLOVER AND M. LAGUNA. *Tabu search*, 1. Kluwer Academic Pub, 1998. 49
- [116] N. SHAKER, M. NICOLAU, G. YANNAKAKIS, J. TOGELIUS, AND M. O'NEILL. **Evolving Levels for Super Mario Bros Using Grammatical Evolution**. In *IEEE Transactions on Computational Intelligence and Games (CIG)*, 2012. 51
- [117] G. SMITH, J. WHITEHEAD, AND M. MATEAS. **Tanagra: A mixed-initiative level design tool**. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, pages 209–216. ACM, 2010. 51
- [118] J. TOGELIUS, R. DE NARDI, AND S.M. LUCAS. **Towards automatic personalised content creation for racing games**. In *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, pages 252–259. IEEE, 2007. 51
- [119] L. CARDAMONE, G. YANNAKAKIS, J. TOGELIUS, AND P. LANZI. **Evolving interesting maps for a first person shooter**. In *Proceedings of the 2011 international conference on Applications of evolutionary computation*, 1 of *EvoApplications'11*, pages 63–72. Springer, 2011. 51
- [120] JULIAN TOGELIUS, MIKE PREUSS, NICOLA BEUME, SIMON WESSING, JOHAN HAGELBÄCK, AND GEORGIOS N. YANNAKAKIS. **Multiobjective Exploration of the StarCraft Map Space**. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, pages 265–272, 2010. 51, 102, 105, 147
- [121] D. ASHLOCK. **Automatic generation of game elements via evolution**. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 289–296, aug. 2010. 51
- [122] M. MATEAS AND A. STERN. **Façade: an experiment in building a fully-realized interactive drama**. In *Game Developers Conference (GDC '03)*, San Jose, CA, USA, March 2003. 52
- [123] Y.G. CHEONG AND R. YOUNG. **Narrative generation for suspense: Modeling and evaluation**. In ULRIKE SPIERLING AND NICOLAS SZILAS, editors, *Interactive Storytelling*, 5334 of *Lecture Notes in Computer Science*, pages 144–155. Springer, 2008. 52
- [124] RICHARD J. GERRIG AND ALLAN B.I. BERNARDO. **Readers as problem-solvers in the experience of suspense**. *Poetics*, 22(6):459 – 472, 1994. Available from: <http://www.sciencedirect.com/science/article/pii/0304422X94900213>. 52
- [125] J. VON NEUMANN AND A.W. BURKS. **Theory of self-reproducing automata**. *IEEE Transactions on Neural Networks*, 5(1):3–14, 1994. 52
- [126] WIKIPEDIA. **Von Neumann neighborhood** — Wikipedia, The Free Encyclopedia, 2011. [Online; accessed 25-July-2011]. Available from: http://en.wikipedia.org/w/index.php?title=Von_Neumann_neighborhood. 53
- [127] WIKIPEDIA. **Moore neighborhood** — Wikipedia, The Free Encyclopedia, 2010. [Online; accessed 25-July-2011]. Available from: http://en.wikipedia.org/w/index.php?title=Moore_neighborhood&oldid=377394151. 53
- [128] J.P. RENNARD. *Collision Based Computing*, chapter Implementation of logical functions in the game of life, pages 419–512. Springer, 2002. 53
- [129] L. JOHNSON, G.N. YANNAKAKIS, AND J. TOGELIUS. **Cellular automata for real-time generation of infinite cave levels**. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, page 10. ACM, 2010. 53, 104
- [130] NATHAN SORENSON AND PHILIPPE PASQUIER. **Towards a Generic Framework for Automated Video Game Level Creation**. In CECILIA DI CHIO, STEFANO CAGNONI, CARLOS COTTA, MARC EBNER, ANIKÓ EKÁRT, ANNA ESPARCIA-ALCAZAR, CHI-KEONG GOH, JUAN MERELO, FERRANTE NERI, MIKE PREUSS, JULIAN TOGELIUS, AND GEORGIOS YANNAKAKIS, editors, *Applications of Evolutionary Computation*, 6024 of *Lecture Notes in Computer Science*, pages 131–140. Springer Berlin / Heidelberg, 2010. 53
- [131] A. LINDENMAYER. **Mathematical models for cellular interactions in development II. Simple and branching filaments with two-sided inputs**. *Journal of theoretical biology*, 18(3):300–315, 1968. 53
- [132] P. PRUSINKIEWICZ AND A. LINDENMAYER. *The algorithmic beauty of plants (The Virtual Laboratory)*. Springer, 1991. 53
- [133] N. CHOMSKY. **Three models for the description of language**. *IRE Transactions on Information Theory*, 2(3):113–124, 1956. 53
- [134] J.E. MARVIE, J. PERRET, AND K. BOUATOUCH. **The FL-system: a functional L-system for procedural geometric modeling**. *The Visual Computer*, 21(5):329–339, 2005. 54
- [135] GLENN MARTIN, SAE SCHATZ, CLINT BOWERS, CHARLES E. HUGHES, JENNIFER FOWLKES, AND DENISE NICHOLSON. **Automatic Scenario Generation through Procedural Modeling for Scenario-Based Training**. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 53(26):1949–1953, 2009. 54
- [136] M.A. BODEN. *The creative mind: myths and mechanisms*. Routledge, 2004. 54
- [137] GRAEME RITCHIE. **Some Empirical Criteria for Attributing Creativity to a Computer Program**. *Minds and Machines*, 17:67–99, 2007. 55
- [138] SIMON COLTON. **Creativity versus the perception of creativity in computational systems**. In *In Proceedings of the AAAI Spring Symp. on Creative Intelligent Systems*, 2008. 55

REFERENCES

- [139] SIMON COLTON AND BLANCA PÉREZ FERRER. **No photos harmed/growing paths from seed: an exhibition.** In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering*, NPAR '12, pages 1–10, Aire-la-Ville, Switzerland, Switzerland, 2012. Eurographics Association. 55
- [140] ALAN TURING. **Digital Computers Applied to Games.** In *Faster Than Thought* (ed. B. V. Bowden), pages 286–295, London, United Kingdom, 1953. Pitman Publishing. 56
- [141] ALAN KOTOK AND JOHN MCCARTHY. *A chess playing program for the IBM 7090 computer.* Master's thesis, Massachusetts Institute of Technology. Dept. of Electrical Engineering, 1962. 56
- [142] ARTHUR SAMUEL. **Some Studies in Machine Learning Using the Game of Checkers.** *IBM Journal*, 3(3):210–229, 1959. 56
- [143] MONROE NEWBORN. *Kasparov vs. Deep Blue: Computer Chess Comes of Age.* Springer, 1997. 56
- [144] CHANG-SHING LEE, MEI-HUI WANG, GUILLAUME CHASLOT, JEAN-BAPTISTE HOOCK, ARPAD RIMMEL, OLIVIER TEYTAUD, SHANG-RONG TSAI, SHUN-CHIN HSU, AND TZUNG-PEI HONG. **The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments.** *IEEE Trans. Comput. Intellig. and AI in Games*, 1(1):73–89, 2009. 57, 119
- [145] J. VON NEUMANN. **Zur Theorie der Gesellschaftsspiele.** *Mathematische Annalen*, 100:295–320, 1928. 58
- [146] GUILLAUME M.J.-B. CHASLOT, MARK H.M. WINANDS, AND H.JAAP HERIK. **Parallel Monte-Carlo Tree Search.** In H.JAAP HERIK, XINHE XU, ZONGMIN MA, AND MARK H.M. WINANDS, editors, *Computers and Games*, 5131 of *Lecture Notes in Computer Science*, pages 60–71. Springer Berlin Heidelberg, 2008. 60
- [147] S. R. K. BRANAVAN, DAVID SILVER, AND REGINA BARZILAY. **Non-Linear Monte-Carlo Search in Civilization II.** In *Proceedings of the International Joint Conference on Artificial intelligence (IJCAI)*, 2011. 60
- [148] C.B. BROWNE, E. POWLEY, D. WHITEHOUSE, S.M. LUCAS, P.I. COWLING, P. ROHLFSHAGEN, S. TAVENER, D. PEREZ, S. SAMOTHRAKIS, AND S. COLTON. **A Survey of Monte Carlo Tree Search Methods.** *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43, March 2012. 60
- [149] LEVENTE KOCIS AND CSABA SZEPESVÁRI. **Bandit Based Monte-Carlo Planning.** In JOHANNES FÜRNKRANZ, TOBIAS SCHEFFER, AND MYRA SPILIOPOULOU, editors, *Machine Learning: ECML 2006*, 4212 of *Lecture Notes in Computer Science*, pages 282–293. Springer Berlin / Heidelberg, 2006. 61
- [150] GUILLAUME CHASLOT, MARK WINANDS, JAAP H. VAN DEN HERIK, JOS UTERWIJK, AND BRUNO BOUZY. **Progressive Strategies for Monte-Carlo Tree Search.** In *Joint Conference on Information Sciences, Heuristic Search and Computer Game Playing Session*, Salt Lake City, 2007. 61
- [151] SYLVAIN GELLY AND DAVID SILVER. **Combining online and offline knowledge in UCT.** In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 273–280, New York, NY, USA, 2007. ACM. 61
- [152] SYLVAIN GELLY AND DAVID SILVER. **Monte-Carlo tree search and rapid action value estimation in computer Go.** *Artificial Intelligence*, 175:1856–1875, July 2011. 61
- [153] PETER AUER, NICOLÒ CESA-BIANCHI, AND PAUL FISCHER. **Finite-time Analysis of the Multiarmed Bandit Problem.** *Machine Learning*, 47:235–256, 2002. 61
- [154] L. J. FOGEL, A. J. OWENS, AND M. J. WALSH. *Artificial Intelligence through Simulated Evolution.* John Wiley, New York, USA, 1966. 62
- [155] SIMON LUCAS. **Evolving Finite State Transducers: Some Initial Explorations.** In CONOR RYAN, TERENCE SOULE, MAARTEN KELIZER, EDWARD TSANG, RICCARDO POLI, AND ERNESTO COSTA, editors, *Genetic Programming, 2610 of Lecture Notes in Computer Science*, pages 241–257. Springer Berlin / Heidelberg, 2003. 62
- [156] LEO BREIMAN, JEROME H. FRIEDMAN, RICHARD A. OLSHEN, AND CHARLES J. STONE. *Classification and regression trees.* Wadsworth International Group, Belmont CA, 1984. 63
- [157] RG DROMEY. **Genetic software engineering-simplifying design using requirements integration.** In *IEEE Working Conference on Complex and Dynamic Systems Architecture*, pages 251–257, 2001. 63
- [158] D. ISLA. **Managing complexity in the Halo 2 AI system.** In *Proceedings of the Game Developers Conference*, 2005. 63
- [159] MCHUGH L. ARGENTON M. DYCKHOFF M. HECKER, C. **Three Approaches to Halo-style Behavior Tree AI.** In *Games Developer Conference, Audio Talk*, 2007. 63
- [160] CHONG-U LIM, ROBIN BAUMGARTEN, AND SIMON COLTON. **Evolving Behaviour Trees for the Commercial Game DEFCON.** In CECILIA CHIO, STEFANO CAGNONI, CARLOS COTTA, MARC EBNER, ANIKÓ EKÁRT, ANNAI. ESPARCIA-ALCAZAR, CHI-KEONG GOH, JUANJ. MERELO, FERRANTE NERI, MIKE PREUSS, JULIAN TOGELIUS, AND GEORGIOS N. YANNAKAKIS, editors, *Applications of Evolutionary Computation*, 6024 of *Lecture Notes in Computer Science*, pages 100–110. Springer Berlin Heidelberg, 2010. 63
- [161] WARREN S. MCCULLOCH AND WALTER PITTS. **A logical calculus of the ideas immanent in nervous activity.** *The bulletin of mathematical biophysics*, 5:115–133, 1943. 63
- [162] T. KOHONEN. *Self-Organizing Maps.* Springer Series in Information Sciences. Springer, 2001. Available from: <http://books.google.dk/books?id=e4ighzyf078C>. 63
- [163] P.J. WERBOS. **Backpropagation through time: what it does and how to do it.** *Proceedings of the IEEE*, 78(10):1550–1560, October 1990. 63
- [164] J J HOPFIELD. **Neural networks and physical systems with emergent collective computational abilities.** *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982. Available from: <http://www.pnas.org/content/79/8/2554.abstract>. 63

REFERENCES

- [165] KENNETH O. STANLEY AND RISTO MIKKULAINEN. **Evolving Neural Networks through Augmenting Topologies**. *Evolutionary Computation*, 10:99–127, 2002. 64, 112, 116
- [166] J. K. OLESEN, G. N. YANNAKAKIS, AND J. HALLAM. **Real-time challenge balance in an RTS game using rt-NEAT**. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pages 87–94, Perth, December 2008. 64
- [167] JON REED, ROBERT TOOMBS, AND NILS AALL BARRICELLI. **Simulation of biological evolution and machine learning: I. Selection of self-reproducing numeric patterns by data processing machines, effects of hereditary control, mutation type and crossing**. *Journal of Theoretical Biology*, 17(3):319 – 342, 1967. 64
- [168] MICHAEL LYNN CRAMER. **A Representation for the Adaptive Generation of Simple Sequential Programs**. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 183–187, Hillsdale, NJ, USA, 1985. L. Erlbaum Associates Inc. 65
- [169] J.R. KOZA. *On the programming of computers by means of natural selection*, 1. MIT press, 1996. 65
- [170] JULIAN MILLER AND PETER THOMSON. **Cartesian Genetic Programming**. In RICCARDO POLI, WOLFGANG BANTZAF, WILLIAM LANGDON, JULIAN MILLER, PETER NORDIN, AND TERENCE FOGARTY, editors, *Genetic Programming, 1802 of Lecture Notes in Computer Science*, pages 121–132. Springer Berlin / Heidelberg, 2000. 65
- [171] MICHAEL THIELSCHER. **The general game playing description language is universal**. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence - Volume Volume Two, IJCAI'11*, pages 1107–1112. AAAI Press, 2011. 67
- [172] NATHANIEL LOVE, TIMOTHY HINRICHS, DAVID HALEY, ERIC SCHKUFZA, AND MICHAEL GENESERETH. **General Game Playing: Game Description Language Specification**, 2008. Available from: http://games.stanford.edu/language/spec/gdl_spec_2004_12.pdf. 68, 109
- [173] ADAM M. SMITH, MARK J. NELSON, AND MICHAEL MATEAS. **Adam M. Smith, Mark J. Nelson, and Michael Mateas**. In *Proceedings of the Conference on Computational Intelligence and Games (CIG)*, pages 91–98. IEEE, August 2010. 68
- [174] ADAM SMITH AND MICHAEL MATEAS. **Variations Forever: Flexibly Generating Rulesets from a Sculptable Design Space of Mini-Games**. In *Proceedings of the Conference on Computational Intelligence and Games (CIG)*, pages 273–280. IEEE, August 2010. 69
- [175] JEFF MALLETT AND MARK LEFLER. **Zillions of Games**, 1998. [Online; accessed 19-September-2011]. Available from: <http://www.zillions.de/>. 69
- [176] CAMERON BROWNE. **Yavalath**, 2007. Available from: <http://www.cameronius.com/games/yavalath/>. 70
- [177] MICHAEL COOK AND SIMON COLTON. **Multi-faceted evolution of simple arcade games**. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, pages 289–296, September 2011. 70
- [178] MICHAEL COOK, SIMON COLTON, AND JEREMY GOW. **Initial results from co-operative co-evolution for automated platformer design**. *Applications of Evolutionary Computation*, 7248:194–203, 2012. 70
- [179] FERDINAND D. SAUSSURE. *Cours de linguistique générale*. Bayot, Paris, 1916. 73
- [180] CHRISTIAN ELVERDAM AND ESPEN AARSETH. **Game Classification and Game Design**. *Games and Culture*, 2(1):3–22, January 2007. 74
- [181] ROBIN MILNER. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, 1st edition, June 1999. Available from: <http://www.worldcat.org/isbn/0521658691>. 84
- [182] FRANK HERBERT. *Dune*. New English Library, 1966. 102
- [183] TOBIAS MAHLMANN, JULIAN TOGELIUS, AND GEORGIOS N. YANNAKAKIS. **Spicing up map generation**. In *Proceedings of the 2012 European conference on Applications of Evolutionary Computation, EvoApplications'12*, pages 224–233, Berlin, Heidelberg, 2012. Springer-Verlag. 102
- [184] JULIAN TOGELIUS, GEORGIOS N. YANNAKAKIS, KENNETH O. STANLEY, AND CAMERON BROWNE. **Search-based Procedural Content Generation: a Taxonomy and Survey**. *IEEE Transactions on Computational Intelligence and AI in Games*, in print:172 – 186, 2011. 104, 133
- [185] JON BENTLEY. **Programming pearls: algorithm design techniques**. *Commun. ACM*, 27:865–873, September 1984. 104
- [186] JON L. NIELSEN AND BENJAMIN F. JENSEN. *Artificial Agents for the Strategy Game Description Language*. Master's thesis, IT University of Copenhagen, 2011. Available from: <http://game.itu.dk/sgdl>. 109
- [187] STUART J. RUSSELL AND PETER NORVIG. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003. 111, 117
- [188] KENNETH H. ROSEN. *Discrete Mathematics and Its Applications*. McGraw-Hill Higher Education, 5th edition, 2002. 115
- [189] MELANIE MITCHELL AND STEPHANIE FORREST. **Genetic algorithms and artificial life**. *Artificial Intelligence*, 1(3):267–289, 1994. 116
- [190] JOHN H. HOLLAND. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press, April 1992. 116, 121
- [191] RAÚL ROJAS. *Neural Networks: A Systematic Introduction*. Springer-Verlag, 1996. 116
- [192] JOHAN HAGELBÄCK AND STEFAN J. JOHANSSON. **Using multi-agent potential fields in real-time strategy games**. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 2, AAMAS '08*, pages 631–638, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems. 120

REFERENCES

- [193] JOHAN HAGELBÄCK AND STEFAN J. JOHANSSON. **The Rise of Potential Fields in Real Time Strategy Bots.** In *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*, pages 42–47, 2008. 120
- [194] S. W. WILSON. **Generalization in the XCS Classifier System.** In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 665–674, 1998. 121
- [195] SIMON M. LUCAS AND JULIAN TOGELIUS. **Point-to-Point Car Racing: an Initial Study of Evolution Versus Temporal Difference Learning.** In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2007. 131
- [196] IBRAHIM OSMAN AND GILBERT LAPORTE. **Metaheuristics: A bibliography.** *Annals of Operations Research*, **63**:511–623, 1996. 133
- [197] T. MAHLMANN, J. TOGELIUS, AND G.N. YANNAKAKIS. **Evolving Card Sets Towards Balancing Dominion.** In *IEEE World Congress on Computational Intelligence (WCCI)*, 2012. 135
- [198] VARIOUS. **The Starcraft Wikia article about the StarCraft II beta test**, October 2010. Available from: http://starcraft.wikia.com/wiki/StarCraft_II_beta. 136
- [199] SAM KASS. **Rock Paper Scissors Lizard Spock.** [Online; accessed 10-October-2012]. Available from: <http://www.samkass.com/theories/RPSSL.html>. 137
- [200] C. SALGE, C. LIPSKI, T. MAHLMANN, AND B. MATHIAK. **Using genetically optimized artificial intelligence to improve gameplaying fun for strategical games.** In *Proceedings of the 2008 ACM SIGGRAPH symposium on Video games*, pages 7–14. ACM, 2008. 142
- [201] C. SALGE AND T. MAHLMANN. **Relevant Information as a Formalised Approach to Evaluate Game Mechanics.** In *Proceedings of the Conference on Computational Intelligence and Games (CIG)*, Copenhagen, DK, August 2010. 142
- [202] A. CINCOTTI AND H. IIDA. **Outcome uncertainty and interestness in game-playing: A case study using synchronized hex.** *New Mathematics and Natural Computation*, **2**:173–181, July 2006. 142
- [203] PHILIP BILLE. **A survey on tree edit distance and related problems.** *Theoretical Computer Science*, **337**(1–3):217 – 239, 2005. 148
- [204] TOBIAS MAHLMANN, JULIAN TOGELIUS, AND GEORGIOS YANNAKAKIS. **Towards Procedural Strategy Game Generation: Evolving Complementary Unit Types.** In *Applications of Evolutionary Computation*, pages 93–102. Springer, 2011. 153, 172
- [205] TOBIAS MAHLMANN, JULIAN TOGELIUS, AND GEORGIOS YANNAKAKIS. **Modelling and evaluation of complex scenarios with the Strategy Game Description Language.** In *Proceedings of the Conference on Computational Intelligence and Games (CIG)*, Seoul, KR, 2011. 156
- [206] HÉCTOR PÉREZ MARTÍNEZ, MAURIZIO GARBARINO, AND GEORGIOS YANNAKAKIS. **Generic Physiological Features as Predictors of Player Experience.** In SIDNEY D’MELLO, ARTHUR GRAESSER, BJÖRN SCHULLER, AND JEAN-CLAUDE MARTIN, editors, *Affective Computing and Intelligent Interaction*, **6974** of *Lecture Notes in Computer Science*, pages 267–276. Springer Berlin / Heidelberg, 2011. 183
- [207] PAOLO BURELLI AND GEORGIOS N. YANNAKAKIS. **Towards Adaptive Virtual Camera Control In Computer Games.** In *International symposium on Smart Graphics*, 2011. Available from: <http://www.paoloburelli.com/publications/burelli2011smartgraphics.pdf>. 183
- [208] HEATHER DESURVIRE, MARTIN CAPLAN, AND JOZSEF A. TOTH. **Using heuristics to evaluate the playability of games.** In *CHI '04 extended abstracts on Human factors in computing systems*, CHI EA '04, pages 1509–1512, New York, NY, USA, 2004. ACM. 183
- [209] DONGSEONG CHOI AND JINWOO KIM. **Why People Continue to Play Online Games: In Search of Critical Design Factors to Increase Customer Loyalty to Online Contents.** *CyberPsychology & Behavior*, **7**(1):11–24, February 2004. 183
- [210] M.A. AHMAD, B. KEEGAN, J. SRIVASTAVA, D. WILLIAMS, AND N. CONTRACTOR. **Mining for Gold Farmers: Automatic Detection of Deviant Players in MMOGs.** In *Computational Science and Engineering, 2009. CSE '09. International Conference on*, **4**, pages 340 –345, aug. 2009. 183
- [211] C. THURAU AND C. BAUCKHAGE. **Analyzing the Evolution of Social Groups in World of Warcraft.** In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 170 –177, aug. 2010. 183
- [212] B.G. WEBER AND M. MATEAS. **A data mining approach to strategy prediction.** In *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 140 –147, sept. 2009. 183
- [213] T. MAHLMANN, A. DRACHEN, A. CANOSSA, J. TOGELIUS, AND G. N. YANNAKAKIS. **Predicting Player Behavior in Tomb Raider: Underworld.** In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, pages 178–185, Copenhagen, DK, August 2010. 183